

# **ISaGRAF**

**Version 3.4**

**РУКОВОДСТВО  
ПОЛЬЗОВАТЕЛЯ**

**CJ INTERNATIONAL**

Information in this document is subject to change without notice and does not represent a commitment on the part of CJ International. 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 CJ International.

© 2000 CJ International. All rights reserved.

Printed in France by CJ International.

2 Rue Hector Berlioz

F-38600 Fontaine

Phone: 33 (0)476 26 87 30

Fax: 33 (0)476 26 87 39

Перевод Науцилус,

119899, Москва,

Воробьевы Горы,

НИИЯФ МГУ, Корпус ВЭ,

Телефон: +7 (095) 939 39 24, +7 (095) 939 58 72

Факс: +7 (095) 939 50 02

E-mail: [root@nautsilus.ru](mailto:root@nautsilus.ru)

<http://www.nautsilus.ru>

ISaGRAF is a registered trademark of CJ International.

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.

# Содержание

<b>A.</b>	<b>РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ .....</b>	<b>A-11</b>
<b>A.1</b>	<b>Первые шаги.....</b>	<b>A-12</b>
A.1.1	Инсталляция ISaGRAF.....	A-12
A.1.2	Использование оперативной информации.....	A-15
A.1.3	Пример приложения.....	A-15
<b>A.2</b>	<b>Управление проектами .....</b>	<b>A-19</b>
A.2.1	Создание проектов и работа с ними.....	A-19
A.2.2	Работа с несколькими группами проектов.....	A-21
A.2.3	Опции .....	A-21
A.2.4	Инструменты .....	A-22
<b>A.3</b>	<b>Управление программами .....</b>	<b>A-23</b>
A.3.1	Компоненты проекта .....	A-23
A.3.2	Работа с программами.....	A-25
A.3.3	Использование средств генерации кода.....	A-28
A.3.4	Другие средства ISaGRAF.....	A-29
A.3.5	Добавление команд в меню Инструменты.....	A-30
A.3.6	Симуляция и отладка работы приложения .....	A-30
<b>A.4</b>	<b>Использование редактора SFC .....</b>	<b>A-33</b>
A.4.1	Основные разделы языка SFC .....	A-33
A.4.2	Создание диаграмм SFC .....	A-35
A.4.3	Работа с существующей диаграммой SFC .....	A-37
A.4.4	Ввод программ 2-уровня .....	A-38
A.4.5	Использование галереи SFC.....	A-42
<b>A.5</b>	<b>Использование редактора потоковых диаграмм.....</b>	<b>A-43</b>
A.5.1	Основы языка FC.....	A-43
A.5.2	Ввод Поточковой Диаграммы.....	A-44
A.5.3	Работа с существующей схемой.....	A-47
A.5.4	Ввод программ уровня 2 .....	A-48

A.5.5	Программирование уровня 2 с помощью Quick LD .....	A-49
A.5.6	Показать опции.....	A-50
<b>A.6</b>	<b>Использование редактора Quick LD .....</b>	<b>A-51</b>
A.6.1	Основы языка LD .....	A-51
A.6.2	Ввод диаграммы LD.....	A-53
A.6.3	Работа с существующей диаграммой.....	A-56
A.6.4	Опции экрана.....	A-57
<b>A.7</b>	<b>Использование редактора FBD/LD .....</b>	<b>A-59</b>
A.7.1	Основы языка FBD/LD.....	A-59
A.7.2	Ввод диаграммы FBD.....	A-61
A.7.3	Работа с существующей диаграммой.....	A-63
A.7.4	Опции изображения .....	A-65
A.7.5	Стили и отслеживание изменений .....	A-65
<b>A.8</b>	<b>Использование текстового редактора.....</b>	<b>A-67</b>
A.8.1	Команды редактирования .....	A-67
A.8.2	Опции .....	A-68
<b>A.9</b>	<b>Дополнительно о программных редакторах .....</b>	<b>A-69</b>
A.9.1	Вызов других инструментов ISaGRAF.....	A-69
A.9.2	Параметры программы .....	A-69
A.9.3	Остальные команды меню «Файл» .....	A-70
A.9.4	Обновление дневника программы .....	A-71
A.9.5	Выбор переменной из словаря .....	A-71
A.9.6	Окно вывода .....	A-72
<b>A.10</b>	<b>Использование редактора словаря .....</b>	<b>A-74</b>
A.10.1	Главное окно словаря .....	A-76
A.10.2	Управление переменными.....	A-77
A.10.3	Описание объектов .....	A-78
A.10.4	Быстрое объявление.....	A-80
A.10.5	Адресная карта Modbus для SCADA.....	A-81
A.10.6	Обмен информацией с другим приложением.....	A-81
<b>A.11</b>	<b>Использование редактора соединений В/В. ....</b>	<b>A-86</b>
A.11.1	Определение плат В/В .....	A-87
A.11.2	Установка параметров плат.....	A-88
A.11.3	Подключение каналов В/В.....	A-88

A.11.4	Непосредственно представленные переменные .....	A-89
A.11.5	Нумерация.....	A-89
A.11.6	Установка индивидуальных защит.....	A-90
<b>A.12</b>	<b>Создание таблиц преобразования.....</b>	<b>A-92</b>
A.12.1	Основные команды .....	A-92
A.12.2	Ввод точек таблицы .....	A-93
A.12.3	Правила и ограничения .....	A-93
<b>A.13</b>	<b>Использование генератора кода .....</b>	<b>A-95</b>
A.13.1	Основные команды .....	A-95
A.13.2	Опции компилятора .....	A-96
A.13.3	Генерация исходных C кодов .....	A-98
A.13.4	Просмотр информации .....	A-99
A.13.5	Определение ресурсов .....	A-99
<b>A.14</b>	<b>Перекрестные ссылки .....</b>	<b>A-105</b>
<b>A.15</b>	<b>Использование графического отладчика .....</b>	<b>A-107</b>
A.15.1	Окно отладчика.....	A-107
A.15.2	Управление приложением .....	A-108
A.15.3	Опции.....	A-110
A.15.4	Команды записи.....	A-111
A.15.5	Изменение по ходу работы .....	A-112
A.15.6	Обмены DDE .....	A-115
<b>A.16</b>	<b>Списки переменных.....</b>	<b>A-117</b>
<b>A.17</b>	<b>Отладка программ ST и IL .....</b>	<b>A-119</b>
<b>A.18</b>	<b>Прожектор.....</b>	<b>A-120</b>
A.18.1	Построение графической схемы .....	A-120
A.18.2	Схема списка .....	A-122
A.18.3	Определение стиля символа .....	A-123
A.18.4	Команды меню Файл.....	A-123
A.18.5	Замечание для пользователей ISaGRAF V3.2 .....	A-124
<b>A.19</b>	<b>Выгрузить .....</b>	<b>A-125</b>
A.19.1	Выгрузка проекта.....	A-125
A.19.2	Параметры связи.....	A-125

A.19.3	Подготовка проекта к выгрузке .....	A-126
A.19.4	Как упакованный исходник запоминается в целевой задаче ....	A-126
A.19.5	Требования к памяти целевой системы .....	A-127
A.19.6	О выгруженном проекте .....	A-127
A.19.7	Совместимость.....	A-127
<b>A.20</b>	<b>Использование средств Диагностики .....</b>	<b>A-128</b>
<b>A.21</b>	<b>Использование симулятора ISaGRAF .....</b>	<b>A-129</b>
A.21.1	Связи с отладчиком .....	A-129
A.21.2	Симуляция В/В .....	A-129
A.21.3	Компоненты библиотеки .....	A-130
A.21.4	Опции.....	A-131
A.21.5	Сохранение и восстановление состояний входов.....	A-131
A.21.6	Профилер цикла.....	A-131
A.21.7	Скрипты симуляции.....	A-132
<b>A.22</b>	<b>Использование менеджера библиотек .....</b>	<b>A-140</b>
A.22.1	Управление элементами библиотеки .....	A-140
A.22.2	Конфигурация В/В .....	A-143
A.22.3	Сложное оборудование В/В .....	A-143
A.22.4	Плата В/В.....	A-144
A.22.5	Функции и блоки, написанные на языке ИЕС .....	A-146
A.22.6	“С” функции и функциональные блоки.....	A-147
A.22.7	Функции преобразования .....	A-148
<b>A.23</b>	<b>Использование утилиты архивирования (Архив).....</b>	<b>A-149</b>
A.23.1	Вызов менеджера архивов.....	A-149
A.23.2	Опции.....	A-150
A.23.3	Дублирование и восстановление .....	A-150
A.23.4	Файлы архива .....	A-150
<b>A.24</b>	<b>Печать готового документа.....</b>	<b>A-152</b>
A.24.1	Настройка таблицы содержания.....	A-152
A.24.2	Опции.....	A-153
<b>A.25</b>	<b>Парольная защита.....</b>	<b>A-156</b>
<b>A.26</b>	<b>Улучшенная техника программирования .....</b>	<b>A-159</b>

A.26.1	Немного больше о средствах ISaGRAF .....	A-159
A.26.2	Заблокированный В/В и виртуальный В/В.....	A-159
A.26.3	Проверка достоверности связи PC-PLC .....	A-162
A.26.4	Каталоги ISaGRAF.....	A-163
A.26.5	Символы приложения.....	A-164
A.26.6	Ограничения ISaGRAF “LARGE” (WDL) workbench .....	A-168

## **В. ОПИСАНИЕ ЯЗЫКА..... В-172**

### **В.1 Архитектура проекта ..... В-173**

В.1.1	Программы.....	В-173
В.1.2	Циклические и последовательные операции.....	В-173
В.1.3	Дочерние SFC программы .....	В-174
В.1.4	Функции и подпрограммы .....	В-174
В.1.5	Функциональные блоки.....	В-176
В.1.6	Язык описания.....	В-177
В.1.7	Правила исполнения.....	В-177

### **В.2 Общие объекты..... В-179**

В.2.1	Основные типы.....	В-179
В.2.2	Константы.....	В-179
В.2.3	Переменные.....	В-181
В.2.4	Комментарии.....	В-184
В.2.5	Макроопределения .....	В-185

### **В.3 Язык последовательных функциональных схем (SFC)..... В-186**

В.3.1	Основной формат схемы SFC .....	В-186
В.3.2	Основные компоненты SFC .....	В-186
В.3.3	Схождения и расхождения .....	В-189
В.3.4	Макро шаги .....	В-192
В.3.5	Действия внутри шагов .....	В-193
В.3.6	Условия, присоединенные к переходам .....	В-198
В.3.7	Динамические правила SFC.....	В-200
В.3.8	Иерархия программы SFC .....	В-200

### **В.4 Язык потоковых диаграмм ..... В-202**

В.4.1	Компоненты FC.....	В-202
В.4.2	Сложные структуры FC.....	В-205
В.4.3	Динамическое поведение FC.....	В-206

В.4.4	Проверка FC .....	B-206
<b>В.5</b>	<b>Язык функциональных блочных диаграмм (FBD) .....</b>	<b>B-207</b>
В.5.1	Основной формат диаграмм FBD .....	B-207
В.5.2	Оператор RETURN .....	B-208
В.5.3	Прыжки и метки .....	B-208
В.5.4	Логическое отрицание .....	B-209
В.5.5	Вызов функций и функциональных блоков из FBD .....	B-209
<b>В.6</b>	<b>Язык релейных диаграмм (LD) .....</b>	<b>B-211</b>
В.6.1	Силовые рельсы и соединительные линии .....	B-211
В.6.2	Множественные соединения .....	B-212
В.6.3	Основные контакты и витки языка LD .....	B-213
В.6.4	Оператор RETURN .....	B-219
В.6.5	Прыжки и метки .....	B-220
В.6.6	Блоки в LD .....	B-221
<b>В.7</b>	<b>Язык структурированный текст (ST) .....</b>	<b>B-223</b>
В.7.1	Основной синтаксис ST .....	B-223
В.7.2	Выражения и скобки .....	B-223
В.7.3	Вызовы функций и функциональных блоков .....	B-224
В.7.4	Специфические булевские операторы языка ST .....	B-225
В.7.5	Основные операторы ST .....	B-227
В.7.6	Расширения ST .....	B-232
<b>В.8</b>	<b>Язык инструкций (IL) .....</b>	<b>B-238</b>
В.8.1	Основной синтаксис IL .....	B-238
В.8.2	Операторы IL .....	B-239
<b>В.9</b>	<b>Стандартные операторы, функциональные блоки и функции</b>	<b>B-</b>
<b>246</b>		
В.9.1	Стандартные операторы .....	B-246
В.9.2	Стандартные функциональные блоки .....	B-266
В.9.3	Стандартные функции .....	B-283

## **С. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ ЦЕЛЕВОЙ ЗАДАЧИ .....**

**C-323**

<b>С.1</b>	<b>Введение .....</b>	<b>C-324</b>
------------	-----------------------	--------------



<b>C.2</b>	<b>Установка.....</b>	<b>C-325</b>
<b>C.3</b>	<b>Работа с целевой задачей ISaGRAF DOS.....</b>	<b>C-326</b>
C.3.1	Работа ISaGRAF: ISA.EXE .....	C-326
C.3.2	Специфические особенности .....	C-327
<b>C.4</b>	<b>Работа с целевой задачей ISaGRAF OS9 .....</b>	<b>C-330</b>
C.4.1	Работа ISaGRAF в однозадачном режиме: isa .....	C-330
C.4.2	ISaGRAF в многозадачном режиме: isaker, isatst, isanet .....	C-331
C.4.3	Специфические особенности .....	C-334
<b>C.5</b>	<b>Работа с целевой задачей VxWorks .....</b>	<b>C-339</b>
C.5.1	Администратор системных ресурсов: isassr.o .....	C-339
C.5.2	Отличительные особенности isa.o, isakerse.o и isakeret.o .....	C-339
C.5.3	Работа ISaGRAF в однозадачном режиме: isa.o .....	C-340
C.5.4	ISaGRAF в многозадачном режиме: isakerse, isakeret .....	C-342
C.5.5	Специфические особенности .....	C-346
<b>C.6</b>	<b>Работа с целевой задачей ISaGRAF NT .....</b>	<b>C-350</b>
C.6.1	Работа ISaGRAF .....	C-350
C.6.2	Общая информация по опциям .....	C-350
C.6.3	Специфические особенности .....	C-354
C.6.4	Интерфейс пользователя .....	C-358
<b>C.7</b>	<b>Программирование на “С” .....</b>	<b>C-364</b>
C.7.1	Обзор .....	C-364
C.7.2	“С” функции преобразований .....	C-365
C.7.3	“С” функции .....	C-370
C.7.4	“С” функциональные блоки .....	C-378
C.7.5	Техника компилирования и линкования .....	C-394
<b>C.8</b>	<b>Связь по Modbus.....</b>	<b>C-401</b>
C.8.1	Сеть MODBUS и протокол.....	C-401
C.8.2	Реализация ISaGRAF.....	C-402
<b>C.9</b>	<b>Управление при отказе питания.....</b>	<b>C-408</b>
C.9.1	Основы.....	C-408
C.9.2	Сохранение переменных приложения.....	C-409
C.9.3	Сохранение состояния программы.....	C-412

<b>C.10</b>	<b>Приложение: Сообщения об ошибках и их описание .....</b>	<b>C-414</b>
<b>D.</b>	<b>ГЛОССАРИЙ.....</b>	<b>D-424</b>
<b>E.</b>	<b>ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ.....</b>	<b>E-432</b>

# **A. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

## A.1 Первые шаги

Этот раздел описывает установку системы разработки ISaGRAF. Кроме того, даны короткие примеры программ ISaGRAF. Они дают пользователю сжатый обзор его основных возможностей и позволяют сразу использовать ISaGRAF.

### A.1.1 Установка ISaGRAF

Этот раздел описывает установку системы ISaGRAF и настройку компьютера для разработки приложений.

#### Программные и аппаратные требования

ISaGRAF может быть установлен на любом компьютере, удовлетворяющем минимальным потребностям Windows 3.1. Однако желательно наличие следующих аппаратных ресурсов:

- Персональный компьютер на базе микропроцессора 80486 или выше (рекомендуется Pentium)
- 8 МБ оперативной памяти (рекомендуется 16 мегабайт)
- Один 3.5" (1.44 МБ) дисковод
- Не менее 20 МБ свободного места на жёстком диске
- Графический адаптер VGA или SVGA и совместимый монитор
- Мышь (необходима для средств графической разработки)
- Параллельный LPT1 порт (необходим для ключа защиты)

До установки ISaGRAF должно быть установлено система Windows:

- Windows 3.1 - расширенная версия для 386
- Windows 95
- Windows NT Version 3.51 или 4.00



#### Использование инсталляционной программы

ISaGRAF устанавливается программой INSTALL. Эта программа копирует программное обеспечение ISaGRAF с диска ISaGRAF на жёсткий диск пользователя. Кроме того, INSTALL добавляет группу ISaGRAF в окно Менеджера Программ и создаёт инициализационный файл «ISA.ini» в поддиректории EXE.

INSTALL - программа Windows, поэтому она должна запускаться из Менеджера Программ Windows или командой Выполнить из меню Пуск Windows 95. Для установки ISaGRAF необходимо выполнить следующие шаги:

- Вставить Диск 1 в соответствующий дисковод
- Из менеджера программ или меню Пуск, запустить "SETUP.EXE" в корневой папке CD-ROM, или "A:\INSTALL.EXE" в случае гибких дисков.

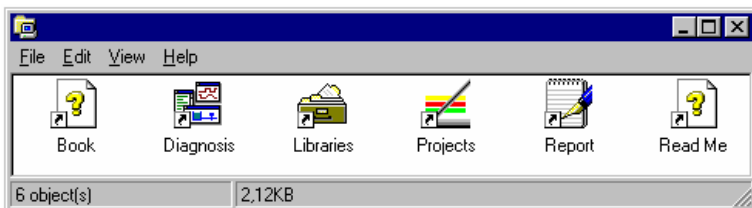
- Следовать соответствующим инструкциям для завершения установки. Рекомендуется устанавливать Систему Разработки ISaGRAF в новой директории, для того чтобы избежать конфликтов с файлами ISaGRAF старых версий.

INSTALL спросит, какие из следующих компонент необходимы:

- Выполняемые программы ISaGRAF
- Оперативно-доступная информация и помощь
- Стандартные библиотеки ISaGRAF
- Примеры программ ISaGRAF

При первой установке ISaGRAF настоятельно рекомендуется включить все компоненты. Позднее, тем не менее, не установленные компоненты могут быть добавлены при реинсталляции ISaGRAF.

По умолчанию имя главной директории ISaGRAF - это «/ISAWIN». Это позволяет легко установить ISaGRAF для Windows на тот же диск, на котором находился ISaGRAF для MS-DOS. Обратитесь к разделу «Директории ISaGRAF» в секции «Улучшенная техника» для более подробной информации о дисковой архитектуре ISaGRAF. Если ISaGRAF был успешно установлен, то следующая группа будет добавлена в Окно Менеджера Программ:



Вот главные пиктограммы ISaGRAF:

- Projects:**.....Управление проектом
- Libraries:** .....Управление библиотекой
- Book:**.....Справочная система ISaGRAF
- Diagnosis:**.....Система диагностики для пользователя
- Read Me:**.....Информация о новой версии ISaGRAF
- Report:**.....Стандартный отчёт об ошибках

В случае если Вы столкнулись с проблемой, используйте стандартную форму отчёта об ошибках. Откройте её, заполните запросы и при помощи команды меню File/Save As сохраните с указанием имени файла. Затем пошлите этот файл CJ International по факсу или по электронной почте.

## — Обновление системных файлов

После завершения установки, необходимо обновить файл CONFIG.SYS до перезагрузки компьютера. Имя директории ISaGRAF не обязательно вставлять в переменную PATH.

ISaGRAF не использует никаких системных переменных DOS. Однако, следующие строки можно добавить в файл CONFIG.SYS:

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

Инструментарий ISaGRAF использует последовательный порт для сообщения с целевой системой PLC. По умолчанию последовательный порт для ISaGRAF - COM-1. Если мышь также использует последовательный порт, выберите COM-2 для мыши. Таким образом, установки по умолчанию будут верны для любых новых программ ISaGRAF.

После изменения файла CONFIG.SYS для вступления изменений в силу необходимо перезагрузить компьютер.

▷

### **Важная информация для пользователей Windows NT:**

При использовании Системы разработки ISaGRAF для Windows NT 3.51 или 4.00 необходимо вставить следующие строки в раздел [WS001] файла ISA.ini в директории \ISAWIN\EXE:

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

Это совершенно необходимо для связи по стандарту RS.

═

### **Защитный Ключ**

Аппаратный защитный ключ предохраняет программное обеспечение ISaGRAF от несанкционированного копирования. Однако, большинство функций ISaGRAF доступно, даже если ключ не установлен. Помимо прочего защитный ключ определяет версию Системы разработки ISaGRAF и максимальный размер разрабатываемого приложения. Если ключ не установлен или установлен неправильно некоторые функции ISaGRAF не будут работать. Это нормальное поведение. Для того чтобы убедиться, что ключ правильно установлен, выберите раздел «**Информация**» меню «**Помощь**» в любом окне ISaGRAF. Доступные опции Системы разработки ISaGRAF будут изображены.

Ключ может быть подключён к любому параллельному порту компьютера. Если в компьютере более одного параллельного порта, то предпочтительнее подключить ключ ISaGRAF и принтер к разным портам. Для некоторых конфигураций компьютер/принтер ключ может не распознаваться, если вывод подключён к независимому (автономному) принтеру. В этом случае отключите принтер, или запустите его в реальном режиме, или перезапустите ISaGRAF.

Следует принять во внимание, что никакого ключа не нужно для ISaGRAF-32.

▷

### **Важная информация для пользователей Windows NT:**

В системах Windows NT должен быть установлен драйвер Sentinel/Rainbow™ для того, чтобы защитный ключ был различим. Прилагается отдельная дискета.

### A.1.2 Использование оперативной информации

Оперативная информация по следующим разделам устанавливается вместе с ISaGRAF:

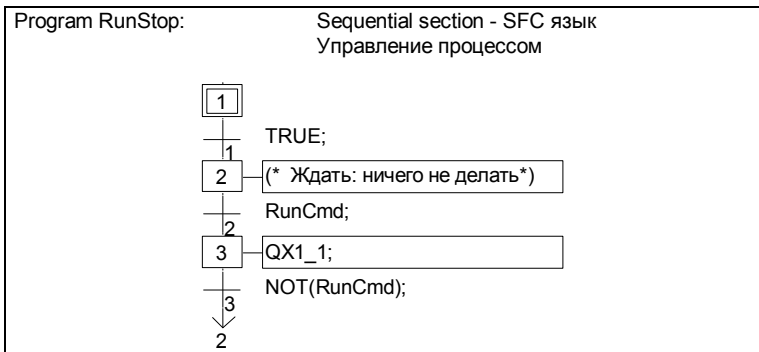
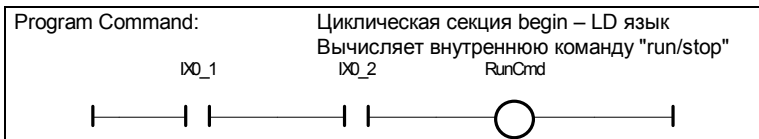
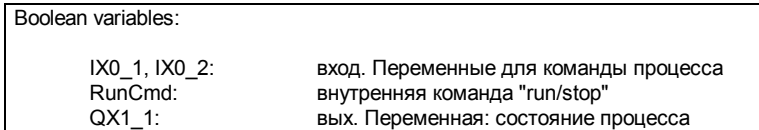
- Описание языка ISaGRAF
- Полное Руководство Пользователя (для любого средства ISaGRAF)
- Технические замечания для элементов библиотеки

Из любого окна ISaGRAF выберите меню «Помощь» для вывода оперативной информации об основных возможностях (таких как языки) и средства ISaGRAF, из которого вызвана справка.

### A.1.3 Пример приложения

Этот раздел объясняет, шаг за шагом, все основные операции, необходимые для создания, разработки и тестирования коротких, но завершённых многоязыковых приложений.

Ниже приведено полное описание такой программы. В ней используется как LD, так и SFC.



Start **Запуск Системы разработки ISaGRAF**

Для запуска Системы разработки ISaGRAF, дважды щёлкните мышью на иконе «Projects» в группе «ISaGRAF». Это приведёт к открытию окна Менеджера Проектов.



### **Создание проекта**

Создайте проект (под названием «RunStop»), используя команду «Новый» из меню «Файл» или кнопку «Новый». В диалоговом окне:

Введите имя проекта: **"RunStop"**.

Выберите конфигурацию ввода-вывода: **"Sim\_Boo"**.

Нажмите кнопку **"Принять"**.

Теперь проект создан.



### **Открытие проекта**

Программы проекта появляются при открытии окна Менеджера Программ ISaGRAF. Пользуйтесь командой «Редактор» окна Менеджера Программ, или щёлкните два раза мышью на имени нужного проекта, или воспользуйтесь кнопкой «Редактор».



### **Создание программ**

Окно Менеджера Программ сейчас открыто и пусто (т.к. ни одна программа не определена). Первая программа создаётся при помощи команды «Новый» меню «Файл» или кнопки «Новый». В окне диалога:

Введите имя программы: **"Command"**.

Выберите язык **"Quick LD"**.

Выберите раздел **"Beginning of cycle"**.

Нажмите кнопку **"Принять"** для создания программы.

Те же операции должны быть произведены для второй программы: Используйте команду «Новый» или кнопки «Новый». В окне диалога:

Введите имя программы: **"RunStop"**.

Выберите язык **"SFC"**.

Выберите раздел **"Sequential"**.

Нажмите кнопку **"Принять"** для создания программы.

Теперь программы созданы. Они появились в окне Менеджера Программ.



### **Объявление переменных**

Перед вводом программы должны быть объявлены внутренние переменные, используемые в данной программе. Это делается при помощи команды «Словарь» меню «Файл» или кнопки Словарь. Переменные ввода-вывода автоматически объявляются при создании проекта.



Теперь открыто окно словаря. При помощи меню **«Файл»**, подменю **«Другой»**, подменю **«Глобальные переменны»** и команды **«Булевские»**, выберите Глобальный Булевский Словарь. С тем же эффектом могут быть использованы кнопки **«Глобальный»** и **«Булевский»**.



Команда **«Новый»** меню **«Редактор»** используется для создания новой булевской переменной. Кроме этого, Вы можете воспользоваться кнопкой



«Вставить объект». В открытом окне диалога введите описание внутренней переменной:

имя: **RunCmd**  
комментарий: **Run/Stop command: internal**  
атрибут: Выберите атрибут "**Внутренняя**"  
Нажмите кнопку «Сохранить»: переменная создана.  
Нажмите кнопку «Отказ» для выхода из окна диалога.

В завершение, покиньте редактор словаря при помощи команды «Выход» меню «Файл». Нажмите «Да» для сохранения изменений.



## Редактирование программы Quick LD

Для начала редактирования LD- программы «Command», дважды щёлкните мышью на её имени в окне Менеджера Программ или воспользуйтесь кнопкой «Редактор».

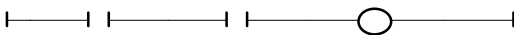


Редактор ISaGRAF Quick LD теперь открыт. Для увеличения рабочей области измените размер окна до полного экрана.

**F2 F3**

Нажмите клавиши F2 и F3:

( ' )



Связь переменных с символами LD: переместите курсор, используя стрелки клавиатуры. Поместите курсор на каждый символ и нажмите клавишу «Enter». Откроется окно диалога выбора переменных.

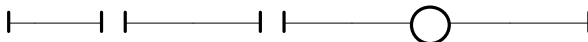
Для первого контакта введите в окно диалога: IX0\_1; затем - «Enter»

Для второго контакта введите в окно диалога: IX0\_2; затем - «Enter»

Для кольца введите в окно диалога : RunCmd ; затем - «Enter»

Программа теперь завершена. Вот полученный результат :

IX0\_1 IX0\_2 RunCmd



Выйдите из редактора и сохраните изменения при помощи команды «Выход» меню «Файл». Щёлкните «Да» для сохранения изменений.



## Редактирование программы SFC

Для начала редактирования программы SFC «RunStop», дважды щёлкните мышью на её имени в окне Менеджера Программ или воспользуйтесь кнопкой «Редактор».



Окно редактора SFC теперь открыто. Для увеличения рабочей области измените размер окна до полного экрана.



Начальный шаг уже существует и выбран. Нажмите стрелку "Вниз", чтобы выбрать пустую ячейку после начального шага (0,1)

**F4 F3**

Нажмите F4, затем F3, чтобы ввести шаг и переход.

**F4 F3**

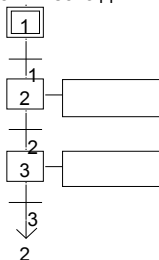
Нажмите F4, затем F3, чтобы ввести еще один шаг и переход.

**F5**

Нажмите F5, чтобы ввести прыжок на шаг и выберите GS2 в качестве назначения прыжка.



Схема готова. Нажмите кнопку "Увеличение" на панели инструментов, чтобы увеличить размер ячеек и дать место для инструкций второго уровня. Вот схема:



Чтобы ввести программирование перехода "2", выберите его, используя стрелки клавиатуры, и нажмите клавишу "Enter". Программное окно уровня 2 открыто. Введите программу уровня 2 для перехода 2:

**RunCmd;**

**^TAB**

Нажмите клавиши "Control + Tab" чтобы передвинуть фокус назад на схему SFC, сдвиньте выбор на шаг 3, и нажмите клавишу "Enter" чтобы редактировать текст уровня 2:

**QX1\_1;**

И сделайте то же самое, чтобы ввести текст перехода 3:

**Not (RunCmd);**

**^F4**

Нажмите клавиши "Control + F4" чтобы закрыть окно уровня 2.

Программа SFC теперь завершена. Выйдите из редактора и сохраните изменения при помощи команды «Выход» меню «Файл». Щёлкните «Да» для сохранения изменений.



### **Создание кода приложения**

Используйте меню «Создать» и команду «Создать приложение» из окна Менеджера Программ для создания кода ( или соответствующую кнопку панели инструментов )

Когда генерация кода завершена, появится окно диалога, которое будет содержать вопрос : закончить генерацию кода *сейчас* ( *now* ) или *продолжить работу* ( *continue* ) : Нажмите кнопку «Выход».




### **Симуляция**

Используйте меню «Отладка» и команду «Симулировать» из окна Менеджера Программ для запуска ядра симулятора ISaGRAF ( то же возможно при помощи соответствующей кнопки панели инструментов ).

При появлении окна симулятора, приложение может быть протестировано. В данном примере, входы 1 и 2 ( зелёные кнопки ) одновременно должны быть закрыты для запуска процесса ( на выходе красный светодиод ).

Закройте окно отладчика для выхода из симулятора : меню «Файл» - команда «Выход»

## A.2 Управление проектами

Для запуска пакета управления проектами ISaGRAF дважды щёлкните мышью на иконе «Проекты» (  ) в группе ISaGRAF. В результате откроется окно «Управление Проектом». Проект, соответствующий одному циклу PLC ( программируемый логический контроллер ) выполняется на целевом PLC. Верхнее окно содержит список существующих проектов. Тестовое описание выделенного проекта находится в нижнем окне.



### **Изменение размеров окна**

Щёлкните мышью на разделителе между списком проектов и их описанием для изменения размера соответствующего окна. Окно описания не может быть полностью спрятано. Оно всегда содержит, по крайней мере, одну строку текста.



### **Вставка разделителей**

Разделительная линия может быть вставлена перед именем любого проекта. Это позволяет группировать проекты, связанные с одним приложением, в списке расположения. Используйте команду «Редактор / Вставка/удаление разделителей» для вставки или удаления разделителя перед выделенным проектом.



### **Перемещение проектов в списке**

Для перемещения проекта в списке нужно, во-первых, выделить его. После этого щёлкнуть мышью на его имени и перетащить в нужное место в списке. При перетаскивании проекта маленькая стрелка слева указывает место, в которое он будет помещён. Кроме этого, Вы можете использовать команду «Редактор» меню «Файл» для перемещения выделенного проекта построчно. Следует иметь в виду, что если разделитель находится перед выделенным проектом, то он перемещается вместе с ним.

## A.2.1 Создание проектов и работа с ними

Команды меню менеджера проектов используются для создания новых проектов, их редактирования и управления существующими.



### **Создание нового проекта**

Для создания нового проекта, введите его имя. В результате этого будет создан пустой проект, который не содержит объектов. Конфигурация ввода-вывода может быть присоединена к созданному проекту. Эта конфигурация должна быть определена в библиотеках. Если конфигурация выбрана, то ISaGRAF автоматически установит соединение ввода-вывода и объявит соответствующие переменные ввода-вывода в словаре нового проекта. Если Вы создаёте или переименовываете проект, то необходимо придерживаться следующих правил относительно его имени :

- Длина имени не должна превышать **8** символов
- Первый символ должен быть **буквой**

- Следующие символы могут быть **буквой**, **цифрой** или символом подчёркивания
  - В имени проекта прописной и строчный регистры не различаются
- После создания проекта, пользуйтесь командой **«Редактор / Текст комментария»** для ввода описания проекта для отображения в списке.



### **Редактирование описания проекта**

Команда **«Проект / Дескриптор проекта»** используется для редактирования описания проекта. Этот документ полностью характеризует проект и его отличия от остальных проектов в списке. Кроме того, он может быть использован для заметок при работе с проектом.



### **Редактирование проекта**

Команда **«Файл / Редактор»** открывает окно менеджера программ для выбранного проекта. Из этого окна можно управлять содержимым проекта ( программами, параметрами приложения и т.д. ). Для редактирования проекта также можно дважды щёлкнуть мышью на его имени.



### **Предыстория изменений**

Система ISaGRAF сохраняет все изменения, относящиеся к компонентам проекта, в файле предыстории. Каждое изменение снабжено именем, датой и временем, когда оно было произведено. Файл предыстории содержит последние **500** изменений. Для каждого проекта существует свой файл предыстории. Файл предыстории - это дополнение к файлу-дневнику данного проекта. Команда **«Проект / История»** позволяет пользователю просмотреть или напечатать предысторию изменений выбранного проекта. Пользователь может выбрать один или несколько элементов в главном списке, нажав соответствующие кнопки :

Принять	.....	закреть окно
Печать	.....	распечатать содержимое списка
Помощь	.....	вывести справку об этом окне диалога
[Удалить] Отмеченные	.....	удалить все выделенные линии из списка
[Удалить] Все	.....	очистить список полностью
Поиск	.....	найти шаблон в списке

Строка ввода над кнопкой **«Поиск»** служит для ввода шаблона поиска. Эта функция не различает регистры. Когда поиск доходит до конца списка, он возобновляется с начала и проходит до начальной позиции.



### **Печать завершённого документа**

Команда **«Проект / Печать»** позволяет пользователю создать и напечатать готовый документ о выбранном проекте. Этот документ может группировать любые компоненты ( программы, переменные, параметры... ) о выбранном проекте. Для создания специального ( не завершённого ) документа, пользователь должен просто определить его содержание.



### **Защита при помощи паролей**

Команда **«Проект / Установить пароль»** позволяет пользователю определить пароль для инструментов и данных выбранного пароля. Обратитесь к разделу **«Защитный пароль»** в конце первой части этого руководства за более

подробной информацией об уровнях защиты и защите данных. Пароль не оказывает влияния на остальные проекты и библиотеки ISaGRAF.

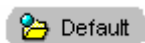
## A.2.2 Работа с несколькими группами проектов

Проектам ISaGRAF соответствует одна директория на диске, в которой хранятся все файлы проектов. "Группа проектов" соответствует списку проектов объединенных вместе в одной и той же корневой директории. Группа проектов идентифицируется по имени. По умолчанию, ISaGRAF создает две группы проектов:

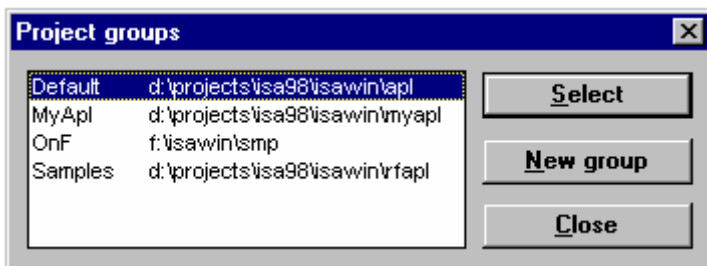
"Default" ..... в "ISAWIN\APL":Ваша рабочая область

"Samples" ..... в "ISAWIN\SMP":примеры приложения поставляемые вместе с системой разработки ISaGRAF

Имя текущей группы проектов написано в панели инструментов, рядом с кнопкой выбора группы проектов:



Вы можете, также, использовать команду "**Файл / Выбрать группу проектов**", чтобы выбрать существующую группу или создать новую. Открывается следующий диалог:



Выберите группу в списке и нажмите "**Выделить**", чтобы активизировать список проектов. Вы можете дважды щелкнуть на имени, чтобы выбрать его. Используйте "**Новая группа**" для создания новой группы. Эта команда может быть использована либо для того, чтобы присвоить имя группы существующей директории, либо чтобы создать новую группу с новой директорией.

Замечание: Группа не может быть выбрана или создана, если открыты другие окна ISaGRAF (менеджер программ, редакторы...).

## A.2.3 Опции

Команды из меню «**Опции**» используются для того, чтобы показать или спрятать инструментальную панель, выбрать шрифт для текста и установить режим «auto»

close» ( автоматическое закрытие ) Менеджера Проектов. Выбранный шрифт используется для изображения описания проектов и в текстовых редакторах ISaGRAF.

При отмене опции **«Держать открытым менеджер проектов»** окно Менеджера Проектов автоматически закрывается при начале работы с проектом.

## **A.2.4 Инструменты**

Команды меню **"Инструменты"** используются для запуска приложений ISaGRAF. Команда **"Инструменты / Архив / Проекты"** запускает менеджер архивов ISaGRAF, чтобы сохранять или восстанавливать проекты. Команда **"Инструменты / Архив / Общие данные "** используется для того, чтобы сохранять или восстанавливать файлы, которые используют все проекты (такие как общие макроопределения).

Команда **"Инструменты / Библиотеки"** запускает менеджер библиотек ISaGRAF в отдельном окне.

Команда **"Инструменты / Импортировать программу IL "** может быть использована для импортирования проекта описанного как отдельная программа IL в текстовом файле, согласно формату файла PLC Open.

## A.3 Управление программами

Окно Менеджера Программ показывает программы приложения и группы ( также называемые модулями или программными блоками ) в виде меню из возможных команд для создания архитектуры проекта, запуска редактора, компилятора и отладчика. Это окно является инструментальным ядром в процессе разработки приложения. Окно Менеджера Программ открывается командой «Открыть» в окне Менеджера Проектов.

### A.3.1 Компоненты проекта

Компоненты проекта называются программами. Программа - это логический объект, который описывает часть выполнения управления. Глобальные переменные ( такие как переменные ввода-вывода ), могут использоваться в любой программе приложения. Локальные переменные могут использоваться только в одной программе. Программы перечислены в **иерархическом дереве**, разделённом на **логические разделы**. Окно показывает программы и связи между ними. Программы «**Верхнего уровня**» появляются слева в иерархическом дереве.

#### ⇒ **Программы верхнего уровня**

Программы верхнего уровня появляются в левой части иерархического дерева. Программы верхнего уровня из первых трёх разделов всегда активны и в течение цикла выполняются в следующем порядке :

- ( Чтение ввода )
- Выполнение программ верхнего уровня раздела **BEGIN**
- Выполнение программ верхнего уровня раздела **SEQUENTIAL**
- Выполнение программ верхнего уровня раздела **END**
- ( Обновление вывода )

Программы разделов «**Begin**» и «**End**» описывают циклические операции. Они не зависят от времени. Программы раздела «**Sequential**» описывают последовательные операции. При этом временные переменные явно фигурируют для разделения основных операций. Главные программы раздела «**Begin**» систематически выполняются в начале каждого рабочего цикла. Главные программы раздела «**Sequential**» в соответствии с главными правилами языка SFC и должны быть написаны на SFC. Программы циклического раздела не могут быть написаны на SFC. Любая программа в любом разделе может иметь одну или более **подпрограмм**.

#### ⇒ **Функции и функциональные блоки**

Программы раздела «**Функции**» могут быть вызваны любой программой из любого раздела проекта. Функция - это алгоритм, который выдаёт одно выходное значение по нескольким входным значениям. Функциональный алгоритм работает только с промежуточными переменными, значение которых стираются между вызовами функции. Это делает невозможным вызов из функции функционального блока. Программы раздела «**Функции**» не могут быть написаны на SFC.

В отличие от функций, **«Функциональные блоки»** ( функциональные блоки ) связаны с алгоритмом, который работает с входными значениями со скрытыми статическими данными, которые копируются системой при каждом вызове функционального блока. Программы из раздела **«Функциональные блоки»** могут быть вызваны любой программой любого раздела проекта. Они не могут быть написаны на SFC.

## ▬ **Подпрограммы**

Подпрограммы - это функции, связанные с одной ( SFC или какой-л. другой ) родительской программой. Подпрограммы могут быть вызваны ( выполнены ) только своей родительской программой. Каждая программа из любого раздела может иметь одну или более подпрограмм. Для написания подпрограмм может использоваться любой язык, кроме SFC.

## ▬ **Дочерние SFC и FC программы**

Дочерняя SFC-программа - это параллельная программа, которая может быть запущена или закрыта своей родительской программой. Родительская и дочерняя программа должны быть написаны на SFC.

Когда родительская программа запускает дочернюю SFC-программу, она помещает SFC-выражения в каждый начальный шаг дочерней программы. Когда родительская программа завершает дочернюю SFC-программу, она очищает все выражения, созданные в ходе работы дочерней.


Любая **FC** программа секции sequential может управлять другими **FC** подпрограммами. Родительская **FC** программа блокируется (ждет) пока исполняется дочерняя FC подпрограмма. Не могут исполняться одновременно операции в родительской FC программе и в одной из ее дочерних FC подпрограмм.

## ▬ **Связь между программы и подпрограммами:**






Подпрограммы и дочерние программы связаны со своими родительскими программами линиями в иерархическом дереве. Связь между SFC-программой и SFC-подпрограммой заканчивается стрелкой. Следует иметь в виду, что такая связь представляет **параллельную** операцию.

## ▬ **Языки программирования**

Каждая программа пишется только на одном **языке**. Этот язык, выбираемый при создании программы, не может быть изменён впоследствии. Однако диаграммы **FBD** могут включать части, написанные на **LD**, а диаграммы **LD** могут содержать вызовы функциональных блоков. Доступные графические языки : **SFC** (последовательные функциональные диаграммы), **FBD** (диаграммы функциональных блоков) и **LD** (релейные диаграммы). Доступные текстовые языки : **ST** (структурированный текст), **IL** (список инструкций). Язык **SFC** зарезервирован для главных и дочерних программ последовательного раздела. Язык каждой программы показан в виде иконы возле имени программы в окне Менеджера Программ. Ниже приведены иконы, используемые для представления языков :

 SFC ..... Sequential Function Chart  
.....(Последовательные Функциональные Схемы)



	FC .....	Flow Chart (Потоковые Диаграммы)
	FBD .....	Functional Block Diagram
		(Функциональные Блочные Диаграммы)
	LD .....	Ladder Diagram (вводятся с помощью редактора Quick LD)
		(Релейные Схемы)
	ST .....	Structured Text (Структурный Текст)
	IL .....	Instruction List (Список Инструкций)

### A.3.2 Работа с программами

Меню **«Файл»** содержит все команды, необходимые для создания, обновления или модификации программ. Она также запускает соответствующие редакторы для ввода содержания прикладных программ.



#### **Создание новой программы**

Функция **«Новый»** меню **«Файл»** позволяет создать программу верхнего уровня, дочернюю или подпрограмму в каждой программной секции. Первая часть информации для ввода - это имя новой программы, подчиняющееся следующим синтаксическим правилам :

- максимальная длина - **8** символов
- Первый символ должен быть **буквой**
- Последующие символы должны быть **буквами, цифрами**, или символами подчёркивания.
- Имена программ не различают регистры

Далее, выберите язык, на котором будет создана новая программа :

SFC .....	Последовательные Функциональные Диаграммы
FC .....	Потоковые Диаграммы
FBD .....	Диаграммы Функциональных Блоков ( могут содержать части LD )
LD .....	Релейные схемы, создаваемые при помощи редактора Quick LD
ST .....	Структурный текст
IL .....	Список Инструкций

Теперь выберите режим выполнения программы :

Begin .....	верхний уровень раздела «Begin».
Sequential .....	верхний уровень раздела «Sequential».
End .....	верхний уровень раздела «End».
Функции .....	в разделе «Функции».
Функциональный блок .....	в разделе «Функциональные блоки».
Дочерняя программа .....	SFC или FC SFC-дочерняя или подпрограмма существующей программы.

При выборе одного из пяти возможных режимов программа размещается в верхнем уровне разделов **Begin**, **End**, **Sequential**, **Функции**. Выбор последнего указывает, что новая программа является SFC-дочерней или FC подпрограммой или подпрограммой. Следует помнить, что программы верхнего уровня должны

быть написаны на SFC или FC и, что SFC и FC не может использоваться для циклических программ и их подпрограмм.



### **Комментарии к программам**

ISaGRAF позволяет присоединить к каждой программе проекта её описание. Этот комментарий изображён мелким шрифтом под её именем. Используйте команду **«Файл / Текст комментария программы»** для ввода или изменения комментария, присоединённого к выделенной программе.



### **Редактирование программы**

Эта команда позволяет изменить содержание программы. Используемый редактор зависит от языка, выбранного для написания программы. Каждая программа редактируется в индивидуальном окне, т.е. возможно редактировать более одной программы в различных окнах. Нажатие клавиши **ENTER** позволяет редактировать выделенную программу. Кроме того, пользователь может щёлкнуть мышью на имени программы для её редактирования.



### **Редактирование файла-дневника**

**Файл-дневник** присоединён к каждой программе. Это текстовый файл, который содержит записи обо всех изменениях программы за время её существования. Этот файл может быть отредактирован, изменён и напечатан в любое время. При выходе из редактора после изменения текста программы автоматически открывается окно ввода записей дневника. Эти записи вставляются вместе с датой и временем и создания в дневник.



### **Словарь переменных**

Команда **«Файл / Словарь»** запускает редактор словаря, в котором объявлены переменные проекта. Переменные могут быть глобальными ( т.е. видимыми в любой программе проекта ) или локальными для выделенной программы. Редактор словаря также может быть использован для объявления **макросов**, которые используются для синонимичной подстановки имён и выражений в текст программы.



### **Параметры функций, функциональных блоков или подпрограмм**

Команда позволяет пользователю определить параметры, передаваемые и возвращаемые выделенной функцией, функциональным блоком или подпрограммой. Эта команда не имеет эффекта, если выделена главная программа разделов **«Begin»** или **«End»** или SFC-программа.

Функции, функциональные блоки или подпрограммы могут иметь до **32** параметров ( входных или выходных ). Функция или подпрограмма всегда имеет ровно один возвращаемый параметр, который должен иметь то же имя, что и функция, для соответствия условиям языка ST.

Список в верхней левой части окна показывает параметры, в порядке способов вызова : сначала параметры вызова, затем - возврата. Нижняя часть окна показывает подробное описание параметра, выделенного в списке. В качестве параметра может быть использован любой тип данных ISaGRAF. Возвращаемый

параметр должен располагаться в списке после параметров вызова. Имена параметров должны подчиняться следующим правилам :

- максимальная длина - 8 символов
- Первый символ должен быть **буквой**
- Последующие символы должны быть **буквами, цифрами**, или символами подчёркивания.
- Имена не различают регистры

Команда **«Вставить»** используется для вставки нового параметра перед выбранным. Команда **«Удалить»** используется для удаления выделенного параметра. Команда **«Упорядочить»** автоматически сортирует параметры так, что возвращаемый параметр становится последним в списке.

## **Перемещение программ в иерархическом дереве**

Команда **«Переименовать»** меню **«Файл»** используется для изменения имени программы или для перемещения её в другой раздел иерархического дерева. Однако язык, на котором написана программа, не может быть изменён. При выполнении этой команды открывается такое же окно, как при создании новой программы и все его поля установлены в соответствии с атрибутами выделенной программы. Имя программы может быть изменено. Также может быть выбран другой раздел дерева или другая родительская программа.

Команда **«Упорядочить программы»** меню **«Файл»** используется для установки корректного порядка программ с одинаковым уровнем и родительской программой. Если выделенная программа - на верхнем уровне, то команда используется для сортировки программ верхнего уровня выделенной секции. Если выделенная программа - на нижнем уровне, то команда используется для сортировки только SFC-программ и подпрограмм, относящихся к той же родительской программе, что и выделенная. После открытия диалогового окна **«Упорядочить программы»** выберите программу, которую Вы хотите переместить, и нажмите кнопку **«Вверх»** или **«Вниз»** для перемещения по списку.



## **Копирование программ**

Для того чтобы скопировать программу, выберите нужную программу из списка и запустите команду **«Файл / Копировать»**. После запуска этой команды появится такое же окно, как при создании программы. Все его поля заполнены атрибутами выделенного файла. Введите имя копии и её месторасположение в соответствующих разделах. Если введённая программа не существует, то она будет создана. Если введённая программа уже существует, то она будет перезаписана. Язык новой программы должен совпадать с языком источника. Нажмите кнопку **«Принять»** для копирования.

Команда **«Копировать в другой проект»** меню **«Файл»** копирует выделенную программу в другой проект с сохранением имени. SFC дочерние и подпрограммы могут быть скопированы вместе с ней. Имена выделенной программы и её дочерних не должны использоваться в проекте, в который производится копирование. Все присоединённые объявления и макросы копируются вместе с программой.



## **Удаление программ**

Для удаления программы выделите её в списке и запустите команду **«Файл / Удалить»**. Программа, имеющая дочерние или подпрограммы не может быть удалена. Дочерние и подпрограммы необходимо удалить раньше. Все логические объявления и макросы удаляются вместе с программой.



## **Импортирование из библиотек**

Команда **«Инструменты / Импорт из библиотеки»** используется для копирования функции или функционального блока, написанного на языке IES, из библиотеки в раздел **«Функции»** или **«Функциональные блоки»** открытого проекта. Локальные переменные и макросы, присоединённые к функции, копируются вместе с ней. Если функция была корректно импортирована из библиотеки, то она может быть помещена в другой раздел иерархического дерева командой **«Файл / Переименовать»**. Для избежания конфликтов имён импортируемая функция должна быть переименована при вставке в проект. Не забывайте также переименовать возвращаемый параметр при импортировании функции.



## **Экспортирование функций или функциональных блоков в библиотеку**

Команда **«Инструменты/Экспортировать в библиотеку»** используется для того, чтобы перенести программу из раздела **«Функции»** или **«Функциональные блоки»** в подходящую библиотеку. Локальные переменные и макросы, присоединённые к экспортируемой функции или функциональному блоку, копируются вместе с ней. Экспортированную функцию или блок следует перекомпилировать в Менеджере Библиотек ISaGRAF, чтобы убедиться в том, что она соответствует параметрам среды библиотеки. Библиотечные функции и функциональные блоки не могут использовать глобальные переменные.

### **А.3.3 Использование средств генерации кода**

Команды меню **«Создать»** используются для генерации кода и для ввода опций и дополнительных данных, используемых при создании кода приложения. Обратитесь к разделу **«Использование генератора кода»** этого Руководства за более подробной информацией об этих средствах.



## **Создание кода приложения**

Команда **«Создать»** запускает генерацию кода проекта. Опции для цели генерации должны быть корректно установлены до запуска команды. До генерации целевого кода любая непроверенная программа проверяется для выявления синтаксических ошибок. ISaGRAF содержит последовательный компилятор, который не перекомпилирует программы, скомпилированные ранее.



## **Проверка выбранных программ**

Команда **«Проверить»** позволяет пользователю проверить синтаксис программ, выделенных в списке. Если при проверке ошибок не обнаружено, то она не будет перепроверяться до тех пор, пока её содержание не будет изменено.



### **Имитация изменения**

Команда «Коснуться» имитирует изменение каждой программы, так что все они будут компилироваться снова при последующей генерации кода.



### **Рабочие опции приложения**

Эта команда открывает окно диалога, в котором вводятся главные параметры работающего приложения. Они включают синхронизацию циклов, управление ошибками, начальный режим и аппаратную реализацию поддерживаемых переменных. Обратитесь к разделу «Использование генератора кода» этого документа за более подробной информацией об этой команде.



### **История изменений**

Эта команда используется для установки опций Генератора Кода ISaGRAF, для создания и оптимизации кода. Обратитесь к разделу «Использование генератора кода» этого документа за более подробной информацией об этой команде.



### **История изменений**

Эта команда открывает окно диалога, в котором изображена история изменений проекта. Обратитесь к разделу «Управление Проектами» этого документа за более подробным описанием этой команды.



### **Определение ресурсов**

«Ресурсы» - это данные, определённые пользователем ( например, файл ), которые должны быть объединены с целевым кодом, так что они могут быть загружены вместе с ним. Обратитесь к разделу «Использование генератора кода» этого документа за более подробным описанием файла определения ресурсов.

## **A.3.4 Другие средства ISaGRAF**

Меню содержит команды, которые вызывают средства ISaGRAF для выделенного проекта. Обратитесь к соответствующим разделам этого документа за более подробным описанием этих средств.



### **Привязка переменных ввода-вывода**

Команда "Соединение В/В" запускает редактор соединения переменных ISaGRAF. Этот инструмент используется для создание связей между объявленными в словаре проекта переменными ввода-вывода и соответствующей аппаратурой.



### **Редактор перекрестных ссылок**

Команда "Перекрестные ссылки" позволяет пользователю определить таблицы преобразования для целого проекта. Таблица преобразования это множество точек, которое определяет множество исправленных значений для аналоговых величин. Преобразование может быть непосредственно присоединено к аналоговым переменным ввода или вывода, позволяя пользователю фильтровать физические величины ввода-вывода или преобразовывать

электрические величины составляющих ввода-вывода в логические значения, значительно упрощая программирование уравнений в приложении.

#### ▬ **Ввод описания проекта**

Команда **«Дескриптор проекта»** используется для редактирования текстового описания проекта. Этот документ полностью характеризует проект. В него можно вносить записи, описывающие изменения в ходе работы с проектом. Он отображается в окне Менеджера Проектов.

#### ▬ **Печать завершённого документа**

Команда **«Печать документа проекта»** позволяет создать и напечатать завершённый документ по выделенному проекту. Этот документ может содержать любые компоненты ( программы, переменные, параметры ) выделенного проекта. Для создания незавершённого документа нужно только определить его содержание.

#### ▬ **История изменений**

Эта команда открывает диалог, в котором представлена история изменений проекта. Смотрите главу "Управление проектами" для более подробных разъяснений.

### **A.3.5 Добавление команд в меню Инструменты**

ISaGRAF обеспечивает способ ввода других команд в меню **"Инструменты"**. Определенные пользователем команды, которые должны быть добавлены, описаны в текстовом файле **"\ISAWIN\COM\ISA.MNU"**. Вы можете добавить до 10 команд. Комментарии, начинающиеся с символа «;», могут быть вставлены в любой строке. Каждая команда описана в двух текстовых строках в соответствии со следующим синтаксисом :

```
M=строка_меню  
C=командная_строка
```

Строка меню - это текст, который должен быть изображён в меню. Командная строка - это команда MS-DOS или Windows, которая может завершаться аргументами. В командной строке Вы можете использовать символ **«%A»** вместо имени открытого проекта и символ **«%P»** вместо имени открытой программы. Следующий пример запускает для редактирования выбранной программы ( для использования с программами IL и ST ) :

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

### **A.3.6 Симуляция и отладка работы приложения**

Команды меню используются для запуска графического редактора ISaGRAF как в режиме симуляции, так и в режиме реального соединения.



## Симуляция

Команда «**Симулировать**» открывает отладчик в режиме симуляции. В этом режиме появляется другое окно, называемое симулятором. Эта программа очень полезна для проверки любой программы, когда целевая машина не доступна. Запуск симулятора закрывает окно Менеджера Программ. Оно снова открывается в режиме отладки, когда открыты одновременно окна отладчика и симулятора. Симулятор. Симулятор не может быть запущен, если открыты дочерние окна ( редактора, генератора кодов, соединения ввода-вывода...). Каждое из них должно быть закрыто до выполнения этой команды. Кроме того, эта команда доступна из меню редакторов ISaGRAF.



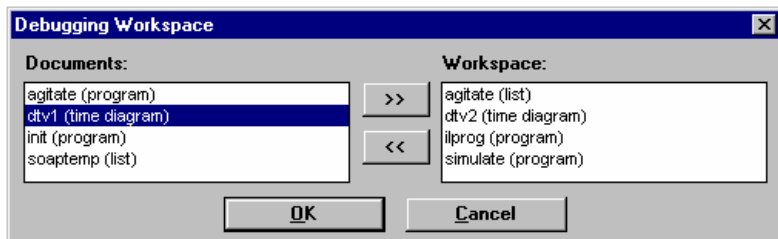
## Реальный отладчик

Команда «**Отладка**» открывает главное окно отладчика и закрывает окно Менеджера Программ. Оно снова открывается после установления связи между отладчиком и целевым приложением. Отладчик не может быть запущен, если целевой код не сгенерирован. Отладчик не может быть запущен, если открыты дочерние окна ( редактора, генератора кодов, соединения ввода-вывода...). Каждое из них должно быть закрыто до выполнения этой команды. Кроме того, эта команда доступна из меню редакторов ISaGRAF.



## Подготовка рабочего пространства отладки

Команда "**Отладка / Рабочее пространство**" позволяет Вам определить список документов для начального рабочего пространства. Такими документами могут быть программы, графика Прожектора, списки переменных. Графики и списки временных диаграмм из предыдущих версий ISaGRAF также записываются документами проекта. Документы, определенные в начальном рабочем пространстве автоматически открываются при запуске симуляции или отладки.



Диалог показывает существующие документы проекта слева и документы, выбранные для начального рабочего пространства. Используйте ">>" и "<<" клавиши для продвижения документов из одного списка в другой. Каждый проект имеет свой собственный список документов для начального рабочего пространства.



## Установка связей

Команда «**Установка связи**» открывает следующее окно диалога. Она позволяет пользователю определить параметры связи, используемые для

коммуникации между отладчиком на главном компьютере и целевой системой ISaGRAF.

**«Номер подчиненного»** определяет целевую систему или задачу ISaGRAF. Он должен находиться в диапазоне от **1** до **255**. Обратитесь к Руководству Пользователя цели для определения используемого номера.

**«Коммуникационный порт»** идентифицирует аппаратное сообщение между инструментарием ISaGRAF и целью. Это может быть имя последовательного порта, или **«Ethernet»**, зарезервированное средство сообщения, использующее «Winsock» версии 1.1.

**«Таймаут»** это время, оставшееся целевой системе для выполнения операций связи между концом запроса отладчик и началом ответа. Оно устанавливается в **миллисекундах**. Поле **«Переповторы»** - это количество автоматических попыток отладчика установить связь перед установлением коммуникационной ошибки.

### ⇒ **Установка последовательной связи**

При выборе последовательного порта ( COM1..4) кнопка **«Установка»** используется для доступа к другим параметрам последовательной связи.

Могут быть выбраны скорость передачи данных, вид контроля и формат. При выборе режима **«аппаратура»** для **«Контроль»** ISaGRAF контролирует линии CTS и DSR для установления подтверждения связи в процессе обмена данными.

### ⇒ **Установка связи Ethernet**

При выборе в качестве последовательного порта «Ethernet», кнопка **«Установка»** для ввода «Адрес интернет» и номера «Номер порта» для связи по протоколу TCP-IP.

Инструментарий использует библиотеку WINSOCK.DLL версии 1.1 для коммуникации по TCP-IP. Этот файл должен быть корректно установлен на жёсткий диск. **«1100»** это номер порта, используемый ISaGRAF по умолчанию.



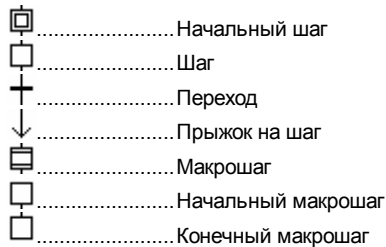
## A.4 Использование редактора SFC

Язык SFC используется для описания операций последовательного процесса. Он использует простое графическое представление для различных шагов процесса и условий, позволяющих изменить активный шаг. Вход в программу SFC осуществляется при помощи графического редактора SFC. SFC - это ядро стандарта IEC 1131-3. Остальные языки обычно описывают действия внутри шагов и логические условия для совершения переходов. Графический редактор SFC позволяет пользователю создавать завершённые программы на SFC. Он совмещает возможности редактирования как текста, так и графики. Таким образом, возможен ввод диаграмм SFC, соответствующих им действий и условий.

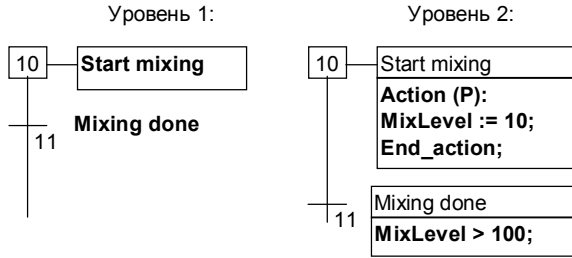
0

### A.4.1 Основные раздела языка SFC

Язык SFC используется для представления последовательного процесса. Он разделяет цикл процесса на несколько строго определённых последовательных **шагов** ( автономных ситуаций ), разделённых **переходами**. Обратитесь к Руководству по Языкам ISaGRAF за более подробной информацией о языке SFC. Компоненты SFC объединены **ориентированными линиями**. По умолчанию, ориентация линии - **сверху вниз**. Вот основные графические компоненты, используемые для построения диаграмм SFC :



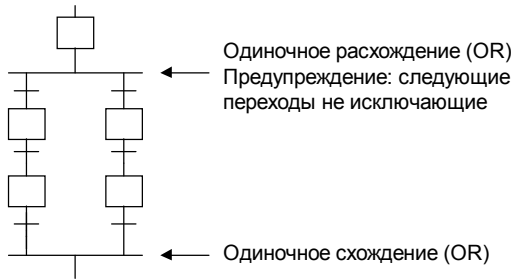
Программирование на SFC обычно разделяется на два различных уровня. **1-Уровень** показывает графические диаграммы, номера ссылок на шаги и переходы и комментарии, присоединённые к ним. **2-Уровень** - это программирование действий внутри шага или условий, присоединённых к переходу, на **ST** или **IL**. **Подпрограммы**, написанные на других языках (**FDB, ST, LD или IL**) могут обращаться к этим действиям или переходам. Ниже приведён пример 1-Уровня и 2-Уровня программирования :



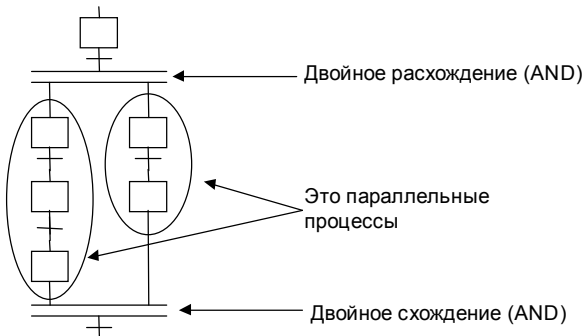
Программа шага на 2 уровне создаётся при помощи текстового редактора. Программа может содержать блоки, написанные на ST или IL. Программа переходов 2 уровня может быть создана на языках ST или IL, или при помощи редактора Quick LD.

## Схождение и расхождение

Используются для представления **множественных связей** между переходами. Простое схождение или расхождение представляет различные **включающие** возможности между различными частями процесса.

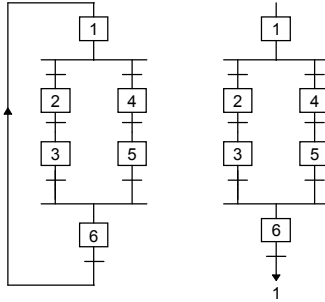


Двойное Схождение представляет **параллельные** процессы.



## ➤ **Переход к шагу**

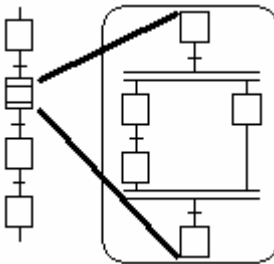
Редактор SFC позволяет пользователю рисовать связи только в направлении **сверху вниз**. **Прыжок** к шагу может быть использован для представления связи с верхней частью диаграммы. Следующие диаграммы эквивалентны :



Прыжок к переходу запрещён и должен представляться двойным (I) Расхождением.

## ➤ **Макро шаг**

Макрошаг - это **уникальное** представление автономной группы шагов и переходов. Макро шаг начинается с **начального шага** и кончается **завершающим шагом**.














Детальное представление шага макроса должно быть описано в той же SFC - программе. Символ шага макроса должен иметь тот же **номер ссылки**, что и начальный шаг макроса. Описание макро шага может содержать другой макро шаг.

### **A.4.2 Создание диаграмм SFC**

Для того, чтобы изобразить диаграмму SFC, пользователю просто нужно ввести основные компоненты диаграммы. Все одиночные линии, соединяющие два элемента ( горизонтальные или вертикальные ) рисуются редактором SFC автоматически. Для вставки символа выберите в редакторе панель инструментов

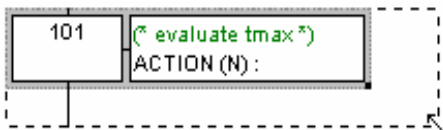
и щёлкните мышью в области диаграммы, в которую нужно произвести вставку. Символы вводятся в текущей позиции. Могут использоваться следующие горячие клавиши:

- F2:  .....Ввести начальный шаг
- F3:  .....Ввести одиночный шаг
- F4:  .....Ввести переход
- F5:  .....Ввести прыжок на шаг
- F6:  F7:  .....Ввести OR расхождение или схождение / Добавить ветви
- †F6:  †F7:  .....Ввести AND расхождение или схождение / Добавить ветви
- F8:  .....Ввести макрошаг
- F9:  †F9:  .....Ввести начальный или конечный шаг для тела макрошага

(Символ "†" определяет комбинацию с клавишей SHIFT)





**Сетка редактирования** показывает центр каждой ячейки редактируемой матрицы. Опции редактора позволяют изображать и убирать сетку во время ввода диаграммы. Сетка очень полезна для изначального расположения диаграммы SFC или выбора её части. Используйте команду «Опции / Настройки» для того, чтобы изобразить или убрать сетку.

Редактор SFC ISaGRAF всегда показывает текущую позицию в матрице. Текущая ячейка отмечена серым. Маленький квадрат в нижнем правом углу может быть использован для свободного изменения размеров ячеек. Отношение X/Y может быть, также изменено.



### Создание Схождения или Расхождения





Схождение или Расхождение всегда изображается **слева направо**. Для изображения Схождения или Расхождения его **левый угол** должен быть помещён в область диаграммы. Тип изображения (одиночный или двойной) устанавливается выбором одной из этих кнопок на панели.

- F6:  F7:  .....Ввести OR расхождение или схождение / Добавить ветви
- †F6:  †F7:  .....Ввести AND расхождение или схождение / Добавить ветви

### Добавление ветви Схождения

Позиции **start** и **stop** каждой **дополнительной ветви** размещаются на линии Схождения или Расхождении при помощи кнопок панели. Левый угол Схождения или Расхождения должен быть выбран перед вставкой новой ветви. Правый угол

имеет тот же стиль ( одиночный или двойной ), что и главный левый угол. Правый угол не может быть вставлен, если не добавлен левый.

F6:  F7:  .....Ввести OR расхождение или схождение / Добавить ветви  
†F6:  †F7:  .....Ввести OR расхождение или схождение / Добавить ветви



### **Вставка макро шага**

Эта кнопка используется для вставки макро шага в главную диаграмму. Тело макро шага должно быть описано где-либо в этой же SFC программе.



### **Тело макрошага**

Макро шаг должен быть описан в той же SFC программе, что и главная диаграмма. Макро шаг должен начинаться **начальным шагом** и кончаться **завершающим шагом**. Часть диаграммы, реализованная как макро, должна быть **автономна**. Начальный шаг макро должен иметь ту же **ссылку**, что и символ макро на главной ветви.

## **A.4.3 Работа с существующей диаграммой SFC**

Для выбора прямоугольной области в схеме Вы можете использовать либо мышку, либо стрелки клавиатуры. Выбранная область помечается серым цветом. Тогда используются команды меню "Редактор":



### **Команды Вырезать / Копировать / Удалить**

Следующие команды доступны в меню «Редактор» при выборе кнопки «стрелка» в панели редактирования :

Вырезать.....переместить выделенный прямоугольник с экрана в буфер SFC

Копировать .....скопировать выделенный прямоугольник с экрана в буфер SFC

Удалить.....Очистить (удалить) выделенный прямоугольник

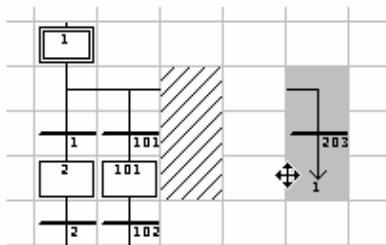
Вставить .....Вставить содержимое буфера SFC в текущей позиции

«Редактор / Копировать» копирует буфер SFC на экран. Команда Копировать / Вставить как с диаграммами SFC, так и с шагами/переходами программ 2-уровня. Кроме того, можно скопировать диаграмму в программе и вставить её в другую SFC программу. Элементы вставляются перед текущей позицией.



### **Сдвиг элементов**

Когда элементы SFC выбраны в схеме, вы можете сдвигать их в другое место путем перетаскивания выбранной области с помощью мыши. Пока вы перетаскиваете выбранную область, начальное положение выбранных элементов закрашивается.



Область назначения должна быть свободна. Во время сдвига SFC символов ввод невозможен.

### ⇒ **Перенумерация шагов и переходов**

Каждый шаг или переход идентифицируется логическим номером в SFC диаграмме. Команда **Редактор/Перенумеровать** позволяет пользователю автоматически установить последовательно нумерацию ссылок для любого шага или перехода в текущей SFC программе. Когда номер шага изменяется, номера всех переходов к этому шагу автоматически обновляются (тоже самое применимо к макро шагам и начальным шагам).



### **Прямой доступ к шагам или переходам**

Команда **"Редактор / Перейти"** позволяет пользователю попадать на существующий шаг или переход. Позиция скроллинга автоматически адаптируется так, чтобы шаг или переход были видны.

### ⇒ **Найти и заменить текст**

Команда **Редактор/Заменить** может быть использована для нахождения или замены текстовых строк в законченной программе (все шаги и переходы). Диалоговое окно **Заменить** используется для ввода искомого текста и открывает уровень 2, в котором находится текст.

## **A.4.4 Ввод программ 2-уровня**

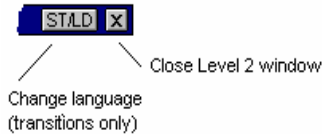
Для ввода программы 2-уровня пользователю нужно дважды щелкнуть мышью на прямоугольнике шага или перехода. Программа 2 уровня появляется справа в окне SFC. Линию раздела между схемой SFC и программой уровня 2 можно свободно перемещать.

Вы можете открыть одновременно 2 области уровня 2. Можно использовать следующие команды клавиатуры, мышки или меню "Редактор":

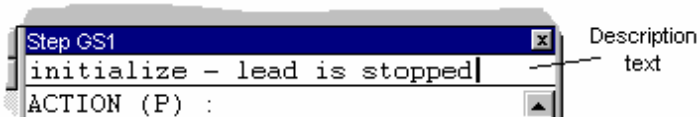
	<i>Клавиатура</i>	<i>Мышь</i>	<i>Меню "Редактор"</i>
Открыть в последнем окне	Enter	Double Click	Редактировать уровень 2
Открыть в отдельном окне	Ctrl+Enter	Ctrl + DoubleClick	Редактировать уровень 2 в отдельном

Когда видны два окна уровня 2, разделение между ними может свободно перемещаться. Кнопка вправо уровня 2 используется для закрытия окна уровня 2.

По умолчанию для написания программ 2-уровня используется язык **ST**. Для переходов 2-уровня может быть использован редактор **Quick LD**. Откроеется независимое окно для ввода программы. Используйте кнопку "**ST/LD**" в полосе заголовка уровня 2 для смены активного языка. Эта команда действительна только тогда, когда окно программирования уровня 2 - пусто.



Прямоугольник редактирования возникает вверху окна уровня 2. Он используется для ввода короткого текста описания. Этот текст появится, как комментарий IEC на чертеже символов SFC. Это очень полезно, так как он может быть использован в других командах, таких, как "Перейти..." и также в надписях шагов и переходов SFC.



Команда "**Опции / Обновить**" может быть использована, когда открыты окна уровня 2, для перерисовки схемы SFC с модифицированными программами уровня 2.



### **Вставка имени переменной**

При программировании на каком-либо языке нажмите эту кнопку для выбора переменной, объявленной в словаре проекта, и вставки ее имени на текущую позицию курсора. При использовании Quick LD нажмите эту кнопку для выбора переменной, которую следует присоединить к выделенному контакту или блоку параметров ввода/вывода.



### **Вставка блока импульсных действий в шаг**

При программировании 2-уровня шага используйте эту кнопку для вставки шаблона блока импульсных действий на текущую позицию курсора. Ниже приведен формат блока импульсных действий :

```

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

```

Импульсное действие - это инструкции, которые исполняются только один раз, когда шаг становится активным. Обратитесь к руководству по языкам ISaGRAF за более подробным описанием программирования на языке SFC.

**N**

### **Вставка блока Не сохранённых действий в шаг**

При программировании шага 2-уровня используйте эту кнопку для вставки шаблона блока не сохранённых действий на текущую позицию курсора. Ниже приведен формат блока:

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

Не сохранённые действия - это инструкции, которые выполняются в каждом цикле PLC при активизации шага. Обратитесь к руководству по языкам ISaGRAF за более подробным описанием программирования на языке SFC.

**P0 P1**

### **Новые P0 и P1 определители действий**

ISaGRAF поддерживает новые определители действий P0 и P1. При программировании 2-уровня шага пользуйтесь этими кнопками для вставки шаблонов блока действий P0 или P1 на текущую позицию курсора. Ниже приведен формат такого блока:

```
Action (P0) :  
    ST statement;  
    ...  
End_Action;  
  
Action (P1) :  
    ST statement;  
    ...  
End_Action;
```

Действия P1 - это инструкции, которые выполняются только один раз при активации шага (также, как и для Импульсного). Действия P0 - это инструкции, которые выполняются только один раз при деактивации шага. Обратитесь к руководству по языкам ISaGRAF за более подробным описанием программирования на языке SFC.

**=**

### **Булевы действия**

Другая текстовая семантика применима непосредственно к действиям над булевыми переменными в соответствии с активностью шага. Эти действия состоят из присоединенных к внутренним булевым переменным или булевым переменным вывода **сигналов активности шага**. Вот синтаксис основных булевых действий:

<boolean_variable> (N);	присвоить переменной сигнал активности шага
<boolean_variable>;	тот же результат ( атрибут N не является необходимым )
/ <boolean_variable>;	присвоить переменной инвертированный сигнал активности шага



Остальные возможности используются для установки или сброса булевой переменной при активации шага. Вот синтаксис таких булевых действий:

<code>&lt;boolean_variable&gt; (S);</code>	установить значение переменной истинным, когда сигнал активности шага становится истинным
<code>&lt;boolean_variable&gt; (R);</code>	установить значение переменной ложным, когда сигнал активности шага становится истинным

## ⇒ **Действия SFC**

Другая текстовая семантика доступна для управления выполнением дочерней SFC программы. Действие SFC - это дочерняя SFC последовательность, начатая или завершенная в соответствии с условиями сигнала активности шага. Действия SFC могут иметь следующие определители: **N** (Не сохраненное), **S** (установить) или **R** (сбросить). Вот синтаксис основных действий SFC:

<code>&lt;child_program&gt; (N);</code>	запускает дочернюю последовательность, когда шаг становится активным. Прекращает выполнение дочерней последовательности, когда шаг становится неактивным.
<code>&lt;child_program&gt;;</code>	тот же результат ( атрибут N не является необходимым)
<code>&lt;child_program&gt; (S);</code>	запускает дочернюю последовательность, когда шаг становится активным. Ничего не делает, когда шаг становится неактивным.
<code>&lt;child_program&gt; (R);</code>	убивает дочернюю последовательность, когда шаг становится активным. Ничего не делает, когда шаг становится неактивным.

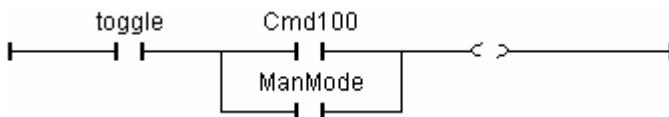
SFC последовательность, объявленная как действие должна быть существующей дочерней программой SFC для редактируемой программы, созданной Менеджером Программ ISaGRAF.

## ⇒ **Переходы, написанные на ST**

Переход 2-уровня - это булево выражение. Для того, чтобы задать его на языке ST. Просто введите булево выражение в соответствии с синтаксисом ST. При желании можно добавить точку с запятой в конце выражения.

## ⇒ **Переходы, написанные на Quick Ladder**

Редактор Quick LD доступен для программирования условий перехода 2-уровня. В этом случае диаграмма состоит из одной ступени с ровно одним витком, представляющим переход. Имя перехода не повторяется на символе витка. Ниже приведен пример условия перехода, написанный на Quick LD:



При программировании на Quick LD используйте стрелки клавиатуры для перемещения по логической сетке и используйте горячие клавиши для вставки символов:

F2:.....вставить контакт после выделенного символа/инициировать ступень

F3:.....вставить контакт перед выделенным символом

F4:.....вставить контакт параллельно выделенному символу

F6:.....вставить блок после выделенного символа

F7:.....вставить блок перед выделенным символом

F8:.....вставить блок параллельно выделенному символу

Кроме этого вы можете щелкнуть мышью на функциональной кнопке на панели внизу окна программы 2-уровня вместо функциональных клавиш.

Нажмите **[RETURN]** при выделенном контакте или параметре блока ввода/вывода для выбора переменной или ввода константного значения. Нажмите **[RETURN]** при выделенном функциональном блоке для выбора его типа. Кроме того вы можете дважды щелкнуть мышью на символе для достижения того же эффекта.

Нажмите **[SPACE]** при выделенном контакте для выбора или изменения его типа (прямой, обратный или Pulse Detection). Обратитесь к разделу «Использование редактора Quick LD» этого документа за более подробным описанием возможностей Quick LD.

#### **A.4.5 Использование галереи SFC**

Редактор ISaGRAF SFC управляет галереей SFC: это набор структур SFC, которые могут быть введены в любую схему SFC. Элементы галереи SFC могут дополнительно включать программы шагов и переходов уровня 2. Используйте следующие команды меню "Инструменты":

**Копировать в галерею SFC** .....копировать выделенные элементы в галерею SFC

**Вставить из галереи SFC** .....вклеить элемент галереи SFC в текущую позицию

При копировании в галерею SFC (т.е. при создании нового элемента галереи SFC), вы можете дополнительно потребовать встроить программу уровня 2 выбранных символов SFC.

## A.5 Использование редактора потоковых диаграмм

Графический редактор потоковых диаграмм ISaGRAF позволяет пользователю вводить программы на языке FC (Flow Chart), с действиями и тестами (решениями) запрограммированными либо на ST, IL либо на языке Quick LD. Потоковые Диаграммы – это диаграммы решений, которые могут быть также использованы для описания последовательных операций, так как они позволяют некоторые дополнительные возможности, такие как неблокированные обратные прыжки.

### A.5.1 Основы языка FC

Потоковые Диаграммы (FC) – это графический язык использующийся для описания последовательных операций. Потоковые Диаграммы состоят из Действий и Тестов. Между действиями и тестами находятся ориентированные связи представляющие потоки данных. Ниже даны графические компоненты языка Потоковых Диаграмм:



**Начало схемы FC:** Символ "Начало" должен возникать в начале программы Потоковых диаграмм. Он уникален и не может быть пропущен. Он представляет начальное состояние диаграммы, когда она активизирована.



**Конец схемы FC:** Символ "конец" должен возникать в конце программы Потоковых Диаграмм. Он уникален и не может быть пропущен. Может быть так, что никакого соединения не подходит к символу "Конец" (всегда виток), но символ "Конец" все же нарисован внизу схемы. Он представляет собой заключительное состояние схемы, когда исполнение было завершено.



**Потоковые связи FC:** Потоковые связи – это линии, которые представляют потоки между двумя точками в диаграмме. Связь всегда заканчивается стрелкой. Две связи не могут исходить из одного источника.



**Действия FC:** Символ действия представляет собой действие, которое нужно выполнить. Действия идентифицируются при помощи числа и имени. Два разных объекта одной схемы не могут иметь одно и то же имя или логический номер. Языками программирования для действий могут быть ST, LD или IL. Действия всегда соединены со связями, одна подходит к нему, другая исходит из него.



**Тесты FC:** Тест представляет собой булевское условие. Тест идентифицируется числом и именем. В соответствии со значением присоединенного выражения на ST, LD или IL, поток направляется либо по пути "Да", либо "Нет". Если программа написана на ST, то за выражением может следовать двоеточие. Если программа написана на LD, то значение условия представляется единственным витком.



**Подпрограмма FC:** Система допускает описание иерархической структуры программ FC. Программы FC организованы в виде иерархического дерева.

Каждая программа FC может вызывать другие FC программы. Такая программа называется дочерней программой программы FC, которая ее вызывает. Программы FC, которые вызывают подпрограммы, называются родительскими программами. Программы FC объединяются вместе в общее иерархическое дерево, используя отношение "предок - наследник". Символ подпрограммы в Поточковой Диаграмме представляет вызов подпрограммы. Исполнение вызывающей программы FC останавливается до завершения работы подпрограммы.



**Специфические действия В/В FC:** Символ специфических действий В/В представляет действие, которое должно быть выполнено. Как другие действия, специфические действия В/В идентифицируются номером и именем. Одна и та же семантика используется в стандартных действиях и специфических действиях В/В. Цель специфических действий В/В состоит только в том, чтобы сделать схему более читаемой и сфокусировать внимание на непереносимых частях схемы. Использование специфических действий В/В – дополнительная особенность. Специфические блоки В/В ведут себя точно так же, как стандартные действия.



**Соединители FC:** Соединители используются для представления связи между двумя точками диаграммы без вычерчивания. Соединитель обозначен как круг и связан с началом потока. Чертеж соединителя завершается, на соответствующей стороне (в зависимости от направления потока данных), с помощью идентификации точки цели (обычно имя символа цели). Соединитель всегда попадает в определенный в Поточковой Диаграмме символ. Символ назначения определяется его логическим номером.



**Комментарии FC:** Блок комментариев содержит текст, который не имеет смысла для семантики схемы. Он может быть введен на любом свободном месте в окне Поточковой Диаграммы, и используется для документирования программ.




## A.5.2 Ввод Поточковой Диаграммы




Чтобы ввести схему, вы должны поместить элементы (действия, тесты решений, соединители...) в графическую область, и начертить потоковые связи между ними.






### Ввод объектов

Чтобы ввести объекты в диаграмму, выберите соответствующую кнопку на панели инструментов и щелкните в графической области, где вы хотите ее ввести. Вы можете либо поместить элемент в свободную область, или ввести его в поток, щелкнув на связи потока. Вставка на связи допускается только для вертикальных связей сверху вниз. Вы можете ввести следующие основные элементы:

- ..... действия, программируемые на ST, IL или Quick LD
- ..... специфические действия ВВ (выделяет непереносимые действия)
- ..... тест (решение), запрограммирован на ST, IL or Quick LD

- ..... соединитель
- ..... вызов подпрограммы FC
- ..... комментарий (текст описания)

Редактор Поточковых Диаграмм ISaGRAF, предлагает вам список классических структур Поточковых Диаграмм. Такие структуры могут быть вставлены только в существующую связь потока. Они не могут быть помещены в пустую область:

- ..... If / Then / Else (двоичный выбор)
- ..... Repeat until (ждать до выполнения условия)
- ..... While (цикл пока выполняется условие)



### **Выбор объектов**

Выбор графических объектов нужен для большинства команд редактирования. Графический редактор ISaGRAF FC позволяет выбрать один или несколько объектов существующих в области диаграммы. Чтобы выбрать объекты, нужно выбрать "**режим выбора**" (кнопка со стрелкой) в панели инструментов. Чтобы выбрать один объект, пользователь должен щелкнуть на его символе.

Чтобы выбрать список объектов, протащите мышку в диаграмме для захвата прямоугольной области. Все графические объекты в прямоугольнике помечаются как "**выбранный**".

Выбранные объекты чертятся темно синим цветом, с маленькими черными квадратами вокруг графического символа. Можно, также удалять или добавлять один объект во множественный выбор щелчком на символе с нажатием клавиш **Shift** или **Ctrl**.

Если делается новый выбор, то все отметки ранее выбранных объектов исчезают. Чтобы удалить существующий выбор, просто щелкните мышкой в пустой области, за пределами прямоугольника ограничивающего выбранные объекты.

Для одиночного выбора, можно использовать стрелки клавиатуры, чтобы передвигать выбор с одного объекта на другой в схеме. Поточковые связи так же могут быть выбраны.



### **Вставка комментариев**

Комментарии могут быть вставлены в любое пустое место диаграммы. Комментарии не оказывают влияние на исполнение программы. Они делают диаграмму более читаемой. Чтобы ввести блок комментариев, выберите соответствующую кнопку в панели инструментов и щелкните на то место в диаграмме, где должен быть помещен комментарий. Щелкните дважды на комментарии, чтобы ввести изменить текст. Никаких специальных ограничивающих символов, типа "(" и ")", не нужно для ввода текста в блоке комментариев. Можно изменить размер блока комментариев путем перетаскивания угла его границы, когда он выбран.



### **Черчение связей потока**

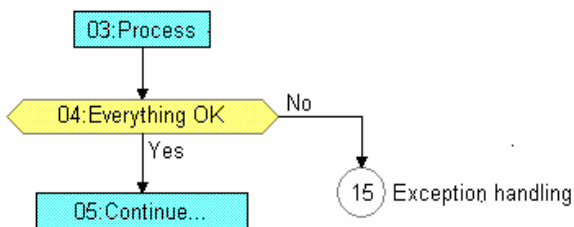
Выберите эту кнопку в панели инструментов, чтобы начертить связь потока между существующими элементами. Связь нужно чертить всегда в направлении потока. Во-первых, выберите несоединенную выходную точку элемента FC, и протащите мышку по направлению к точке назначения, чтобы ввести связь. Точкой назначения может быть либо верх (входная точка) несоединенного элемента FC, либо любое место в существующей связи. Точки схождения связей показаны маленькими серыми кружками на Потокковой Диаграмме. Точки схождения могут быть также выбраны или сдвинуты для того, чтобы упорядочить диаграмму.



## Использование соединителей

Редактор Потокковых Диаграмм ISaGRAF позволяет использовать графические соединители, как замену видимых связей потока. Соединители могут быть очень полезны при использовании их вместо очень длинных линий и повышают читаемость схемы. Соединитель не может быть использован для установки связи с другой программой FC.

Соединитель помещается на схеме как другой объект FC. Он представляется в виде кружка содержащего числовую ссылку элемента (назначения связи потока). Краткое описание элемента появляется рядом с кружком соединителя.



## Передвижение объектов

Чтобы передвинуть объекты в схеме, вы должны выбрать их и перетащить мышку, чтобы передвинуть их в схеме. Вы можете передвинуть один элемент или множественный выбор. Элементы не могут накладываться друг на друга при передвижении. Перемещение элементов нельзя использовать для соединения их с существующей связью.

Когда один элемент (действие, тест...) перемещается, редактор Потокковых Диаграмм ISaGRAF автоматически двигает вместе с выбранным элементом все элементы находящиеся ниже и соединенные с ним. Эта особенность не работает в случае множественного выбора.



## Изменение размеров объектов

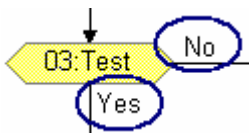
Можно свободно изменять размер любого графического объекта, за исключением символов "Начало", "Конец" и соединителей. Чтобы изменить размер элемента вы, во-первых, должны его выбрать. Затем перетащите мышкой маленький квадратик, нарисованный на его границе, чтобы изменить размер.

Когда элемент соединен со связью потока, изменение его горизонтальных размеров действует на правую и на левую границы, так что при изменении размеров элемент остается правильно центрированным на связи.



### **Обмен выходов на тесте**

Вы можете обменять положения выходов Да / Нет на тесте (решение). Чтобы сделать это, просто, щелкните дважды либо на "Да" либо на "Нет" возле символа теста.



## **A.5.3 Работа с существующей схемой**

Команды меню **"Редактор"** используются для изменения или завершения существующей диаграммы. Большинство из этих команд действуют на элементы, выбранные в настоящее время на диаграмме.

### **Исправление схемы**

Клавиша DEL может быть использована для удаления выбранных элементов. Всякие связи удаляются вместе с выбранными элементами. Используйте команду **"Редактор / Отменить"** для восстановления элементов после команды DEL. Команда DEL может быть также применена к группе элементов выбранных в диаграмме. Команды **"Вырезать"**, **"Копировать"**, **"Вставить"** меню **"Редактор"** используются для перемещения или копирования выбранных элементов.

### **Вставить и заменить**

Команды **"Редактор / Найти Заменить"** могут быть использованы для поиска и замены текстовых строк во всей программе (все действия и тесты запрограммированные на ST, IL или Quick LD). Диалог Найти/Заменить используется для ввода текста поиска и для прямого открытия секции программы, где находится текст.



### **Прямой доступ к элементу**

Команда **"Редактор / Перейти"** дает пользователю прямой доступ к графическому элементу, находящемуся в схеме. Позиция скроллинга автоматически изменяется так, чтобы элемент был виден. Достигнутый элемент выделяется.



### **Перенумерация элементов**

Команда **Редактор/Перенумеровать** используется для перенумерации элементов потоковой диаграммы. Каждый элемент FC помещенный на схему идентифицируется логическим номером в FC диаграмме. Номера размещаются редактором каждый раз, когда вводится новый элемент. **"Перенумеровать"**

позволяет перенастроить нумерацию элемента в соответствии с его положением в схеме. Номера увеличиваются сверху вниз и слева направо

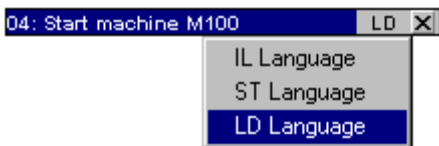
## A.5.4 Ввод программ уровня 2

Для ввода программы 2-уровня пользователю нужно дважды щелкнуть мышью на символе действия или теста. Программы уровня 2 показаны справа в окне FC. Разделительная линия между схемой FC и программами уровня 2 может свободно двигаться. Вы можете открыть одновременно одну или две области уровня 2. Имеются следующие команды клавиатуры, мышки и меню "Редактор":

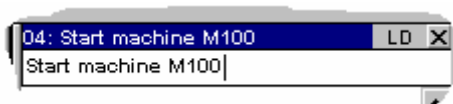
	<i>Клавиатура</i>	<i>Мышь</i>	<i>Меню "Редактор"</i>
Открыть в последнем окне	Enter	Double Click	Редактировать уровень 2
Открыть в отдельном окне	Ctrl+Enter	Ctrl + DoubleClick	Редактировать уровень 2 в отдельном окне

Когда видны два окна уровня 2, разделение между ними может быть свободно передвинуто. Кнопка справа на полосе заголовка уровня 2 используется для закрытия окна уровня 2.

По умолчанию для написания программ 2-уровня используется язык **ST** (Структурный текст). Языком программирования может быть также **IL** или **Quick LD**. Имя выбранного языка появляется в маленьком прямоугольнике в полосе заголовка уровня 2. Запустите команду "**Опции / Установить язык уровня 2**" из меню или щелкните по этому прямоугольнику, чтобы изменить активный язык. Эта команда действительна только, если окно программ уровня 2 пусто.



Прямоугольник редактирования возникает сверху окна уровня 2. Он используется для ввода короткого текста описания. Этот текст, также, появится, как комментарий IEC, на чертеже символов FC. Это очень полезно, так как это используется другими командами, такими как "Перейти..." и также в надписях действий и тестов FC.

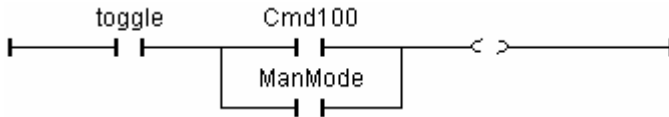


Команда "**Опции / Обновить**" может быть использована в любой момент, когда открыто окно уровня 2, чтобы обновить основную схему FC измененными программами уровня 2.



## A.5.5 Программирование уровня 2 с помощью Quick LD

Quick LD доступен для программирования 2-уровня. В случае теса решения LD диаграмма состоит из одной ступени с ровно одним витком, представляющим решение. Имя решения не повторяется на символе витка. Ниже приведен пример теста, написанный на Quick LD:



При программировании на Quick LD используйте стрелки клавиатуры для перемещения по логической сетке и используйте следующие горячие клавиши для вставки символов:

- F2:..... вставить контакт после выделенного символа
- F3:..... вставить контакт перед выделенным символом
- F4:..... вставить контакт параллельно выделенному символу
- F5:..... добавить виток параллельно выделенному символу (не для тестов)
- F6:..... вставить блок после выделенного символа
- F7:..... вставить блок перед выделенным символом
- F8:..... вставить блок параллельно выделенному символу
- F9:..... добавить символ прыжка в параллель с выбранным витком (не для тестов)

Прыжок ведет к имени ступени. Имя ступени может быть введено нажатием ENTER, когда выбран заголовок ступени. Редактор ISaGRAF помнит уже введенные метки ступеней, определены ли они для имени ступени или для операции прыжка. Диалог "Прыжок/Метка" дает вам возможность ввести новую метку или выбрать существующую. Если вы вводите новое имя, оно будет автоматически добавлено в список. Кнопка "**Уничтожить**" используется для удаления выделенного имени из списка. Она не удаляет метку на ступени, которую вы выбрали в диаграмме. Чтобы это сделать, просто нажмите **Принять** когда прямоугольник редактирования пуст.

Кроме этого вы можете щелкнуть мышью на функциональной кнопке на панели внизу окна программы 2-уровня вместо функциональных клавиш.

Нажмите ENTER при выделенном контакте или параметре блока ввода/вывода для выбора переменной или ввода константного значения. Нажмите ENTER при выделенном функциональном блоке для выбора его типа. Кроме того вы можете дважды щелкнуть мышью на символе для достижения того же эффекта.

Нажмите Control + SPACE при выделенном контакте для выбора или изменения его типа (прямой, обратный). Обратитесь к разделу «Использование редактора Quick LD» этого документа за более подробным описанием возможностей Quick LD.

## A.5.6 Показать опции

Команда "**Опции / Настройки**" открывает диалог, где сгруппированы все параметры и опции, касающиеся рабочего пространства редактора и чертежа диаграммы. Используйте проверки группы "Рабочее пространство" чтобы показать или скрыть панель инструментов и статусную панель редактора. Опции группы "Document" позволяют вам показать или скрыть точки решетки редактирования и показать схему либо черно-белой, либо цветной.



Используйте кнопку "Увеличение" панели инструментов, для изменения текущего масштаба. Эта команда всегда имеется при работе с программой Quick LD присоединенной к действию или тесту.



Используйте кнопку "Сетка" панели инструментов чтобы показать или скрыть точки решетки редактирования. Эта команда, также, имеется при работе с программой Quick LD присоединенной к действию или тесту.

Используйте команду "**Опции / Шрифт**" для выбора имени шрифта символа который используется во всех документах ISaGRAF. При вызове из блоков ST или IL, вы можете определить размер шрифта. При выборе шрифта для графического вида (FC или Quick LD), стиль шрифта и размер не относятся к делу и их не нужно выбирать. Графические редакторы ISaGRAF всегда рассчитывают размер шрифта в соответствии с текущим масштабом.

## A.6 Использование редактора Quick LD

Язык LD допускает графическое представление булевых выражений. Булевы операторы AND, OR, NOT явно представлены в топологии диаграммы. Булевы выходные переменные присоединены к виткам графика. Редактор Quick LD допускает простой ввод диаграмм при помощи клавиатуры или мыши. Элементы автоматически связываются и размещаются по ступеням самим редактором. Пользователю не нужно изображать никаких соединений. Кроме этого, редактор Quick LD размещает ступени диаграммы таким образом, что место, занимаемое диаграммой, всегда оптимально.

### A.6.1 Основы языка LD





Программа на языке LD представлена списком **ступеней**, на которых расположены контакты и витки. Ниже приведены основные компоненты LD диаграммы :

#### **Головная часть ступени ( левая шина питания )**

Каждая ступень начинается с левой шины питания, начальное значение которой истинно. Редактор Quick LD автоматически создаёт левую шину питания, когда первый контакт ступени создан пользователем. Каждая ступень может иметь логическое имя, которое может использоваться в качестве метки для инструкций прыжка.






#### **Контакты**

Контакты изменяют поток булевых данных в соответствии со значением булевой переменной. Имя переменной изображено над символом контакта. Следующие типы контактов поддерживаются редактором Quick LD :

-  ..... прямой контакт
-  ..... инвертированный контакт
-  ..... контакт с определением положительного (переднего) фронта
-  ..... контакт с определением отрицательного (заднего) фронта

#### **Витки**

Виток представляет действие. Состояние ступени (состояние связи в левой части витка) используется для изменения булевой переменной. Имя переменной изображено над символом витка. Следующие типы витков поддерживаются редактором Quick LD :

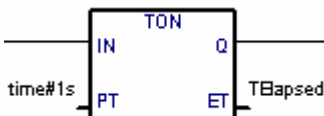
-  ..... прямой виток
-  ..... инвертированный виток
-  ..... "установить" виток действия
-  ..... "сбросить" виток действия
-  ..... виток с определением положительного (переднего) фронта

 ..... виток с определением отрицательного (заднего) фронта



## Функциональные блоки

Блок диаграммы LD может представлять функцию, функциональный блок, подпрограмму или оператор. Его первый входные и выходные параметры всегда соединены со ступенью. Другие параметры описаны вне прямоугольника блока.



## Конец ступени ( правая шина питания )

Ступени заканчивается правой шиной питания. Редактор Quick LD автоматически вставляет правую шину питания, когда пользователь помещает виток в диаграмму.



## Символ прыжка

Символ прыжка всегда ссылается на метку ступени, т.е. имя этой ступени, определённое где-либо в этой же диаграмме. Он помещается в конце ступени. Если состояние ступени истинно, выполнение диаграммы прямо перейдёт к помеченной ступени. Следует иметь в виду, что обратные прыжки опасны, так как они могут привести к блокированию цикла PLC в некоторых случаях.



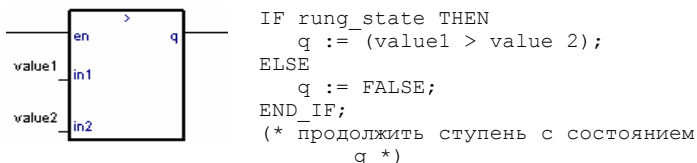
## Символ возврата

Символ возврата размещается в конце ступени. Он означает, что выполнение программы должно быть остановлено, если значение ступени истинно.



## Ввод «EN»

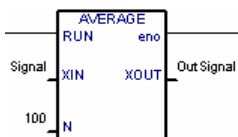
В некоторых операторах, функциях и функциональных блоках первый ввод не имеет булевского типа данных. Так как первый ввод всегда должен быть соединен со ступенью, то другой ввод автоматически вставляется на первую позицию, называемую «EN». Блок выполняется только в том случае, если ввод «EN» истинен. Ниже приведён пример оператора сравнения и его эквивалент на языке ST :



## Вывод «ENO»

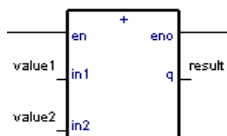
В некоторых операторах, функциях или функциональных блоках первый вывод не имеет булевского типа данных. Поскольку первый вывод всегда должен быть соединён со ступенью, другой вывод автоматически вставляется на первую

позицию. Он называется «**ENO**». Этот вывод всегда находится в том же состоянии, что и первый вход блока. Ниже приведён пример функционального блока AVERAGE и его эквивалент на языке ST :



```
AVERAGE(rung_state, Signal, 100);
OutSignal := AVERAGE.XOUT;
eno := rung_state;
(* продолжить шину с состоянием eno
  *)
```

В некоторых случаях одновременно необходимы как **EN**, так и **ENO**. Ниже приведён пример арифметического оператора и его эквивалент на ST :



```
IF rung_state THEN
  result := (value1 + value2);
END_IF;
eno := rung_state;
(* продолжить шину с состоянием eno
  *)
```

## ИИИ Ограничения редактора Quick LD

Редактор ISaGRAF Quick LD не позволяет продолжить ступень ( вставить новый контакт или виток) вправо от витка. Если одной ступени требуется сразу несколько выводов, то соответствующие витки должны быть параллельны.

### А.6.2 Ввод диаграммы LD

Все команды редактирования редактора Quick LD могут быть выполнены как при помощи клавиатуры, так и мыши.



#### Сетка редактирования

Диаграмма LD вводится при помощи логической матрицы. Каждая ячейка матрицы может содержать до одного символа LD. Пользуйтесь стрелками клавиатуры или мышью для перемещения по ячейкам. Текущая ячейка помечена инвертированным цветом. Для некоторых операций по копированию/вырезанию/вставке можно помечать сразу несколько ячеек. При помощи мыши это делается так : просто перемещайте курсор мыши внутри диаграммы. При помощи клавиатуры, пользуйтесь стрелками при нажатой клавише SHIFT.

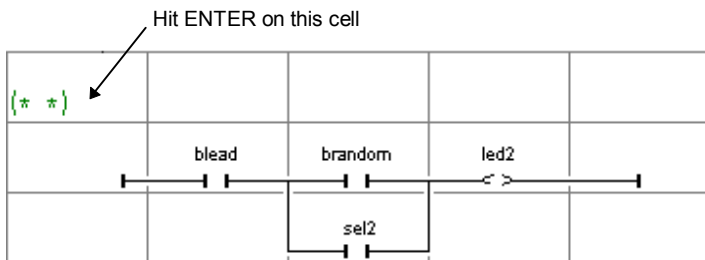
#### Вставка новой ступени

Для вставки новой ступени в диаграмму, переместите область выбора за последнюю существующую ступень и вставьте контакт ( нажмите F2 или соответствующую кнопку на панели ). Появится новая ступень с одним витком и одним контактом.

#### Вставка комментария для ступени

Каждая ступень может сопровождаться документацией размером до двух строк текста. Для ввода комментария переместите выделенную область до ячейки над

ступенью и нажмите клавишу ENTER или дважды щёлкните мышью на этой ячейке :



### Ввод метки ступени

Каждая ступень может быть идентифицирована при помощи имени. Это имя может быть использовано в качестве целевой метки для операции перехода. Для ввода или изменения метки ступени переместите выделенную область на начало и нажмите ENTER или дважды щёлкните мышью на ячейке :



Редактор ISaGRAF Quick LD запоминает ранее введённые имена ступеней и адреса переходов. Окно диалога "Прыжок/Метка" даёт Вам возможность выбрать существующую метку или создать новую.

Если Вы введёте новое имя, оно будет автоматически добавлено в список. Кнопка «Уничтожить» используется для удаления выделенного имени из списка. Она не удаляет метку на выделенной ступени диаграммы. Для этого просто нажмите **Принять**, когда окно диалога пусто.

### Вставка символов в ступень

Вставка символов ( контактов, колец, блоков... ) в существующую ступень всегда производится в соответствии с текущей выделенной областью экрана. Вам нужно выделить требуемую ячейку внутри ступени одну из следующих функциональных клавиш для вставки :

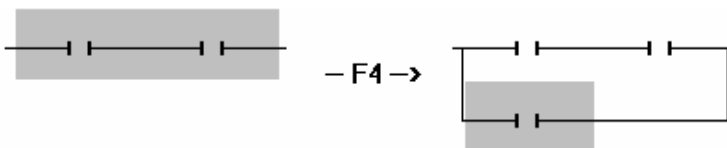
- F2.....контакт слева от выделенного символа
- F3.....контакт справа от выделенного символа
- F4.....контакт параллельно выделенному символу

F6.....блок слева от выделенного символа  
 F7.....блок справа от выделенного символа  
 F8.....блок параллельно выделенному символу

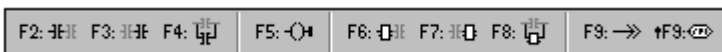
Следующие команды разрешены, когда выделенная область находится на выходе ступени (витке) :

F5.....добавить виток параллельно выделенному  
 F9.....добавить символ перехода параллельно выделенному  
 Shift + F9 .....добавить символ возврата параллельно выделенному

При параллельной вставке ( F4/F8 ), если выбраны несколько контактов ступени, символ вставляется параллельно группе выделенных элементов. Ниже приведён пример :



Для вставки символов в диаграмму, Вы также можете воспользоваться командами меню **«Вставить»**. При помощи мыши, Вы можете щёлкнуть на кнопке панели внизу экрана, на которой изображён нужный тип символа :



## Ввод символов

Для того, чтобы связать символ переменной с контактом или витком, выделите его и нажмите ENTER. Дважды щёлкните мышью на контакте или витке. Появится окно выбора переменной. Обратитесь к разделу **«Дополнительные сведения о редакторах программ»** этого документа за более подробной информацией об использовании этого окна. Для связи функции, функционального блока или оператора с блоком, нажмите ENTER, когда выделенная область находится внутри прямоугольника. Для связи символа переменной с параметром входного или выходного блока, выделенная область должна располагаться соответственно, **вне** прямоугольника блока.

Для ввода текста, обычно, используются диалоги, включающие списки блоков или переменных. Если выбран режим **"Ручной ввод с клавиатуры"** в меню **"Опции"**, символы переменных и имена вводятся прямо через прямоугольник редактирования текста. Введите новый текст и нажмите клавишу **"Enter"**, чтобы установить его, или **"Escape"**, чтобы отказаться от изменений и закрыть редактор. Прямоугольник редактирования текста в режиме **"ручной ввод с клавиатуры"** не может быть закрыт мышкой.



## Изменение типа контакта или витка

Команда **«Редактор / изменить виток /тип контакта»** изменяет тип выделенного контакта или витка. Контакт может быть прямым, инвертированным, с

положительным или отрицательным определением фронта. Виток может быть прямым, инвертированным, с положительным или отрицательным определением фронта. Нажатие пробела приводит к тому же эффекту.

### ⇒ **Вставка ступени в диаграмму**

Команда **«Редактор / Вставить ступень»** вставляет новую ступень в диаграмму перед выделенной. Вставленная ступень имеет один контакт и один виток.

## **A.6.3 Работа с существующей диаграммой**

Команды меню **«Редактор»** используются для изменения или дополнения существующей диаграммы. Большинство этих команд имеют дело с выделенными элементами диаграммы.

### ⇒ **Исправление диаграммы**

Клавиша DEL может быть использована для удаления выделенных элементов. Невозможно удалить виток, переход или символ возврата, если это единственный выход ступени. Используйте команду **«Редактор / Отменить»** для восстановления элементов, удалённых командой DEL. Команда DEL может быть применена, как к одному выделенному элементу, так и к их группе. Также этой командой можно удалять комментарии. При удалении заголовка ступени, вся ступень удаляется целиком.

### ⇒ **Копирование символов**

Команды **«Вырезать»**, **«Копировать»**, **«Вставить»** меню **«Редактор»** используются при перемещении или копировании выделенных элементов. Эти команды не оперируют с комментариями. Команда **«Редактор / Специальная вставка»** даёт Вам выбор для вставки элементов :

- слева от выделенного элемента
- справа от выделенного элемента
- параллельно выделенному элементу

### ⇒ **Управление целой ступенью**

Все команды редактирования ( Удалить, Копировать, Вырезать ) оперируют со ступенью целиком, если выделен её заголовок ( левая шина питания ). Это обеспечивает простоту перемещения ступеней в диаграмме, просто перемещением выделения на первую колонку. Кроме того, можно расширить область выделения по вертикали, чтобы она включала в себя несколько заголовков ступеней. В этом случае команды редактирования применимы к списку ступеней.

### ⇒ **Поиск и замена**

Команды меню **«Редактор / Найти»**, **«Редактор / Заменить»** используются для нахождения и замены текста в диаграмме. Могут быть найдены только полные имена. Поиск производится по контактам, виткам, именам блоков, параметрам блоков и меткам ступеней. Он не может быть использован для нахождения строки в комментарии. Команда Заменить не может быть использована для изменения типа блока. Поиск может быть осуществлён вверх или вниз от выбранной



позиции. Он заканчивается по достижении конца диаграммы. Для поиска доступны следующие горячие клавиши

**ALT+F2** найти следующий элемент с тем же именем переменной, что и выбранный. Этот поиск может быть применен только к функциональным блокам и меткам ступеней.

**ALT+F5** найти следующий виток с тем же именем переменной, что и выбранный. Этот поиск, в основном, используется в режиме отладки для быстрого поиска ступеней, влияющих на переменную.

#### A.6.4 Опции экрана

Команды меню «**Опции**» используются для изменения способа изображения диаграмм LD, а также для того, чтобы показать или убрать некоторый тип информации.

##### ☰ **Комментарии ступеней**

Используйте команду "Опции / Ступень (комментарии)" для того, чтобы спрятать или показать комментарии ступеней на диаграмме в целом. Это может понадобиться для более компактного изображения большой диаграммы, так как комментарий занимает целую строку редактируемой матрицы. Эта опция может быть изменена в любое время.

##### ☰ **Имена и псевдонимы**

Каждая переменная, связываемая с контактом, витком или параметром блока ввода-вывода идентифицируется символьным именем. Кроме этого, редактор Quick LD допускает представление «**псевдонима**» для каждой переменной. Псевдоним переменной - это комментарий, усеченный перед первым символом « : » и ограниченный 16 символами. Ниже приведены примеры :

<i>комментарии переменной:</i>	<i>псевдонимы:</i>
short text	short text
long text with no separator	long text with n
short text: long description	short text

Псевдонимы не влияют на выполнение диаграммы и должны рассматриваться как комментарии с синтаксической точки зрения. Псевдоним автоматически извлекается из комментария, когда переменная выбирается в списке. Он не может быть изменен вручную. Используйте команду "Опции / Контакты и витки" для выбора режима изображения идентификаторов переменных. Возможны следующие режимы :

- отображать только имена переменных
- отображать только псевдонимы переменных
- отображать одновременно имена и псевдонимы переменных

Редактор Quick LD не изменяет автоматически LD документы, когда псевдонимы переменных изменены в словаре. Используйте команду "Опции / Контакты и витки / Изменить псевдонимы", чтобы изменить все псевдонимы в редактируемой диаграмме. Вы можете также установить опцию "Всегда

изменять по открыть " из "Опции / Контакты и витки ", чтобы запросить ISaGRAF автоматически изменять все псевдонимы каждый раз, когда открывается программа Quick LD. Предупреждение: Установка этой опции может значительно увеличить время открытия программы.



### **Опции изображения**

Команда "Опции / Настройки" открывает диалог, где объединены все параметры и опции, касающиеся рабочего пространства редактора и чертежа графической диаграммы LD.

Используйте окна метки в группе «Рабочее пространство» для того чтобы показать или спрятать панели редактора. Опции группы «Документ» позволяют показать или спрятать сетку редактирования, а также разрешают/запрещают цветное редактирование.



Опции группы «Увеличение» позволяют Вам выбрать главный коэффициент увеличения. С той же целью может быть использована кнопка «Увеличение» панели редактирования.



Также Вы можете изменить отношение X/Y в сетке редактирования. Это часто делается для уменьшения ширины ячейки при работе с короткими именами. С той же целью используется кнопка «ширина ячейки» панели редактирования.

Используйте команду "Опции / Шрифт" для выбора имени шрифта символа который используется во всех документах ISaGRAF. При выборе шрифта для графического вида, стиль шрифта и размер не относятся к делу и их не нужно выбирать. Графические редакторы ISaGRAF всегда рассчитывают размер шрифта в соответствии с текущим масштабом.

## A.7 Использование редактора FBD/LD

Графический редактор ISaGRAF FBD/LD позволяет пользователю создавать конечные программы FBD, которые могут содержать части, написанные на LD. Он сочетает в себе возможности редактировать как текст, так и графику. Таким образом, можно создавать диаграммы и соответствующие им входы и выходы. Поскольку этот редактор предназначен для языка FBD, то чистые LD диаграммы предпочтительнее создавать при помощи редактора Quick LD.

### A.7.1 Основы языка FBD/LD

Язык **FBD** - это графическое представление многих различных типов равенств. **Операторы** представлены функциональными прямоугольниками. Функциональные входы присоединяются к левой части прямоугольника. Функциональные выходы присоединяются к правой части. Входы и выходы диаграммы (**переменные**) соединены с функциональными прямоугольниками при помощи **логических связей**. Выходы одного функционального прямоугольника могут соединяться со входами другого.

Язык **LD** допускает графическое представление булевых переменных. Логические операции **AND, OR, NOT** полностью представлены в топологии диаграммы. Булевы входные переменные присоединяются к графическим **контактам**. Булевы выходные переменные присоединяются к графическим **виткам**. Контакты и витки соединяются друг с другом и с левыми и правыми шинами питания при помощи **горизонтальных линий**. Каждый отрезок линии имеет булево состояние - **истинное** или **ложное**. Булево состояние одинаково для всех отрезков, соединённых вместе. Любая горизонтальная линия, соединённая с левой **вертикальной шиной питания**, имеет **истинное** состояние.

Диаграммы всегда интерпретируются слева направо и сверху вниз. Обратитесь к Руководству по Языкам ISaGRAF за более подробной информацией по языкам LD и FBD. Вот основные графические компоненты языков LD и FBD, поддерживаемые редактором FBD/LD :



#### ***Левая шина питания***

Слева ступени должны быть присоединены к **левой шине питания**, которая имеет начальное истинное состояние. Кроме того, редактор позволяет присоединять любые булевы переменные к левой шине питания.



#### ***Правая шина питания***

Вита могут быть соединены с **правой шиной питания**. Это дополнительная возможность редактора. Если виток не соединен ни с чем справа, то он включает правую шину питания в своё собственное изображение.



#### ***Вертикальное соединение LD «OR»***

Вертикальное соединение LD допускает несколько соединений слева и справа. Каждое соединение справа эквивалентно комбинации левых соединений, связанных логическим ИЛИ.



## Контакты

Контакты изменяют поток булевых данных в соответствии со значением булевой переменной. Имя переменной изображено над символом контакта. Следующие типы контактов поддерживаются редактором FBD/LD :

- ..... прямой контакт
- ..... инвертированный контакт
- ..... контакт с определением положительного (переднего) фронта
- ..... контакт с определением отрицательного (заднего) фронта



## Витки

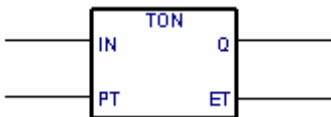
Виток представляет действие. Состояние ступени ( состояние связи в левой части витка ) используется для изменения булевой переменной. Имя переменной изображено над символом витка. Следующие типы колец поддерживаются редактором FBD/LD :

- ..... прямой виток
- ..... инвертированный виток
- ..... "установить" виток действия
- ..... "сбросить" виток действия
- ..... виток с определением положительного (переднего) фронта
- ..... виток с определением отрицательного (заднего) фронта



## Функциональные блоки

Блок диаграммы FBD может представлять функцию, функциональный блок, подпрограмму или оператор. Его входные и выходные параметры должны быть соединены с контактом или витком или другими входами или выходами блока. Формальные параметры описаны внутри прямоугольника блока.



## Метки

Метки могут быть помещены в любом месте диаграммы. Метка используется как цель для инструкций прыжка, т.е. для изменения порядка выполнения диаграммы. Метки не соединяются с другими элементами. Для повышения читаемости диаграммы метки рекомендуется размещать в её левой части.



## Прыжок

Символ прыжка всегда ссылается на метку, расположенную где-либо в диаграмме. Слева он должен быть соединён с булевым элементом. Если левое соединение истинно, то выполнение программы сразу переходит к помеченному участку. Следует иметь в виду, что обратные прыжки опасны, так как в некоторых случаях приводят к зацикливанию.



### **Символ возврата**

Символ возврата соединяется с булевым элементом. Он означает, что выполнение программы должно быть остановлено, если значение ступени истинно.



### **Переменные**

Переменные в диаграмме представлена внутри маленького прямоугольника, соединённого слева или справа с другим элементом диаграммы.



### **Связи соединения**

Связи соединения изображаются между элементами диаграммы. Они всегда изображаются от входа к выходу ( в направлении потока данных ).



### **Связи соединения булева отрицания**

Некоторые булевы связи представлены маленьким кругом в своём правом окончании. Это представляет булево отрицание информации, проходящей через связь.



### **Углы, определяемые пользователем**

Пользователь может определять точки на линии связи. Они позволяют вручную управлять направлением связи. Если не задано никакого угла, то ISaGRAF по умолчанию использует свой алгоритм управления.

## **А.7.2 Ввод диаграммы FBD**

Для ввода диаграммы Вам нужно поместить элементы ( блоки, переменные, контакты, вита ) в графическую область и изобразить связи между ними.



### **Вставка объектов**

Для вставки объекта в диаграмму, выберите соответствующую кнопку на панели и щёлкните мышью в области, в которую нужно произвести вставку.



### **Выделение объектов**

Выделение графических объектов в основном используется для команд редактирования. Редактор ISaGRAF FBD/LD допускает выбор одного или нескольких существующих объектов диаграммы. Для выделения объектов используется кнопка **«выбрать»** ( отмеченная стрелкой ) на панели инструментов. Для выделения одного объекта пользователю нужно просто щёлкнуть мышью на его символе. Для выделения нескольких объектов, переместите мышь, выделив прямоугольную область диаграммы. Все графические объекты, входящие в эту область будут отмечены как **выделенные**.

Выделенный объект изображается с маленьким чёрным квадратом вокруг своего символа. При выполнении новой операции выделения выделенные ранее объекты перестают быть выделенными. Для снятия выделения, просто щёлкните мышью в пустой области вне выделенного прямоугольника.



### **Вставка комментариев**

Комментарии могут быть вставлены в любое место диаграммы. Комментарии не влияют на выполнение программы, они повышают её читабельность. Для вставки блока комментариев, выберите эту кнопку на панели и с помощью мыши выделите прямоугольник на экране, в который нужно вставить комментарий. После этого введите текст комментария. Никакие специальные открывающие или закрывающие символы, такие как «(\*)» или «(\*)», не нужны. Размер блока комментариев можно изменить, перетаскивая мышью угол его границы при выделении.



### **Перемещение объектов**

Для перемещения объекта в диаграмме Вам нужно выделить его и перетащить в нужное место диаграммы. Для перемещения связанных объектов пользователю нужно просто переместить их символы. Редактор автоматически перерисует связи между перемещёнными объектами.



### **Изображение связей**

Выберите одну из этих кнопок на панели для изображения связи между точками соединения существующих элементов. Если Вы изобразите связь от точки соединения до пустого места, то она будет автоматически закончена углом, определяемым пользователем. Таким образом, Вы можете продолжить изображение другим отрезком.



### **Изменение изображения связей**

Команда «Инструменты / Двигать линию» используется, когда выделенная связь в диаграмме меняет свой автоматический тип управления. Эта команда не влияет на связи, соединённые с углом, определяемым пользователем. Если связь изображена тремя отрезками, то эта команда изменит положение второго отрезка. Ниже приведён пример.



### **Изменение типа связи**

Вы можете легко изменить тип связи (возможно, с булевым отрицанием) дважды щёлкнув мышью на её правом крае.



### **Изображение ступеней LD**

Для изображения новой ступени LD сначала вставьте новую шину питания. Затем вставьте виток: он будет автоматически связан с шиной питания. Остальные

контакты и вертикальные ИЛИ соединения могут быть непосредственно вставлены в линию ступени без изображения новых линий связи.

Когда новый LD контакт или виток вставлен в пустую область редактируемой диаграммы, новая горизонтальная ступень автоматически изображается от нового вставленного элемента до существующих шин питания слева и справа. Этого не происходит автоматически, если новый элемент вставляется не между шинами питания. Вставленные контакт или виток могут быть перемещены внутри созданной ступени. Горизонтальные линии, созданные редактором во вставке, могут быть выделены и удалены. Вы можете ставить новый LD контакт или символ вита на горизонтальной линии существующей ступени. Редактор автоматически изменит размер ступени и соединит со вставленным элементом.



### **Множественное соединение**

Множественное соединение может быть создано справа от любой точки **вывода**. Это означает, что информация **распространяется** в несколько различных точек диаграммы. То же свойство распространяется на каждый правый край. Число линий, изображённых справа от точки соединения, не ограничено. Две линии соединения не могут быть соединены справа с одной и той же точкой **входа**. Исключением являются следующие символы LD :



.....правая шина питания



.....множественное соединение слева (ИЛИ)

Символы LD могут иметь неограниченное число входов.

## **А.7.3 Работа с существующей диаграммой**

Команды меню **“Редактор”** используются для изменения ли завершения существующей диаграммы. Большинство из этих команд оперируют с выделенными элементами.



### **Исправление диаграммы**

Клавиша DEL может быть использована для удаления выделенных элементов. Связи удаляются вместе с выделенными элементами. Используйте команду **«Редактор / Отменить»** для восстановления элементов, удалённых командой DEL. Команда DEL применима к группе выделенных элементов. Команды **«Вырезать»**, **«Копировать»**, **«Вставить»** меню **«Редактор»** используются для перемещения и копирования выделенных элементов.



### **Поиск и Замена**

Команды меню **«Редактор / Найти»**, **«Редактор / Заменить»** используются для нахождения и замены текста в диаграмме. Могут быть найдены только полные имена. Поиск производится по контактам, виткам, именам блоков, параметрам блоков и меткам. Он не может быть использован для нахождения строки в комментарии. Команда Заменить не может быть использована для изменения имени блока. Поиск может быть осуществлён вверх или вниз от выбранной позиции. Он заканчивается по достижении конца диаграммы.



### **Отображение порядка выполнения**

Когда FBD диаграмма содержит обратный цикл, порядок исполнения не может быть представлен стандартным методом слева направо / сверху вниз. Во избежание путаницы, используйте команду **«Инструменты / Показать порядок выполнения»** или комбинацию клавиш **Control + F1** для изображения порядка выполнения, который будет использоваться при компиляции. Ярлыки, пронумерованные от 1 до N располагаются возле символов, вызывающих действия (витки, установка значений переменных, функциональные блоки).



### **Ввод символов и текста**

Дважды щёлкните мышью на элементе для ввода соответствующего символа или текста. Это применимо к переменным, контактам и виткам, текстам комментариев и меткам. При использовании этого метода с контактом или витком это также позволяет изменить их тип ( прямой, обратный...).

Для ввода текста, обычно, используются диалоги, включающие списки блоков или переменных. Если выбран режим **"Ручной ввод с клавиатуры"** в меню **"Опции"**, символы переменных и имена вводятся прямо через прямоугольник редактирования текста. Введите новый текст и нажмите клавишу **"Enter"**, чтобы установить его, или **"Escape"**, чтобы отказаться от изменений и закрыть редактор. Прямоугольник редактирования текста в режиме "ручной ввод с клавиатуры" не может быть закрыт мышью.

Если в меню **«Опции»** установлен режим **«Автоввод»**, то символ переменной должен вводиться непосредственно каждый раз при вставке нового контакта или витка. При вставке переменной или метки необходимо вставить и их символ.



### **Выбор типа функционального блока**

Дважды щёлкните мышью на блоке для изменения его типа. Тип блока выбирается из списка доступных операторов, функций и функциональных блоков. Кроме того, команда позволяет изменить количество входных параметров в случае коммуникативного оператора ( т.е. AND, OR, ADD, MUL... )



### **Получение свободного пространства**

Когда вы нажимаете правую клавишу мышки в области чертежа FBD, появляется всплывающее меню. Оно содержит следующие команды, которые могут быть использованы для ввода и удаления свободного пространства в положении курсора мышки:

**Вставить ряд:.....** Эта команда вводит свободное пространство, состоящее из 4 рядов в соответствии с шагом решетки, начиная с позиции курсора мышки, где открылось всплывающее меню.

**Удалить ряд:.....** Эта команда удаляет неиспользованное горизонтальное пространство (ряды), начиная с позиции курсора мышки, где открылось всплывающее меню. Эта команда не может быть использована для удаления элементов FBD.

Когда всплывающее меню открыто, серая линия на чертеже FBD показывает, где пустое пространство будет вводиться или удаляться.



## A.7.4 Опции изображения

Команды меню «**Опции**» используются для изменения изображения диаграммы FBD на экране.

### ☰ **Изменение расположения**

Команда "**Опции / Настройки**" открывает диалог, где сгруппированы все параметры и опции, касающиеся рабочего пространства редактора и чертежа диаграммы. Используйте проверки группы "Рабочее пространство" чтобы показать или скрыть панель инструментов и статусную панель редактора. Опции группы "Document" позволяют вам показать или скрыть точки решетки редактирования.



Используйте кнопку "Увеличение" панели инструментов, для изменения текущего масштаба. Эта команда всегда имеется при работе с программой Quick LD присоединенной к действию или тесту.

Используйте команду "**Опции / Шрифт**" для выбора имени шрифта символа, который используется во всех документах ISaGRAF. При вызове из блоков ST или IL, вы можете определить размер шрифта. При выборе шрифта, стиль шрифта и размер не относятся к делу и их не нужно выбирать. Графические редакторы ISaGRAF всегда рассчитывают размер шрифта в соответствии с текущим масштабом.

## A.7.5 Стили и отслеживание изменений

Редактор ISaGRAF LD/FBD позволяет вам приписать графический стиль любой компоненте диаграммы LD/FBD. Стиль, в основном, определяется, как специальная окраска диаграммы. Но ISaGRAF, также, использует стили, чтобы разрешить отслеживание изменений в диаграммах.

Заметим, что стили не видны во время симуляции или отладки, так как цвета (красный и голубой) используются в этом режиме для высвечивания TRUE / FALSE состояний переменных.

### ☰ **Предопределенные стили**

Следующие стили предопределены:

**Нормальный**.....Чертеж по умолчанию (черный). Для отслеживания изменений, стиль "нормальный" определяет элементы, имеющие этот стиль, как часть первоначальной диаграммы. Элементы стиля "Нормальный", обычно сканируются во время исполнения.

**Модифицир**.....Элементы помеченные "модифицированный" окрашены розовым. Для отслеживания изменений, стиль "модифицированный" используется, чтобы высветить элементы, которые были добавлены или изменены со времени появления первоначальной диаграммы. Элементы стиля " модифицированный ", обычно сканируются во время исполнения..

**Удаленный**.....Элементы помеченные "удаленный" окрашены серым, с пунктирной линией. Такие элементы не принимаются во внимание во время исполнения диаграммы. Этот стиль используется для отслеживания удаленных элементов.

**Адаптирован**.....В дополнение к предопределенным стилям, редактор ISaGRAF LD/FBD позволяет вам выбрать любой цвет, примененный к части диаграммы. Такие элементы рассматриваются, как имеющие стиль "Адаптированный". Использование стиля "Адаптированный" не влияет на выполнение диаграммы.

Используйте команды подменю **"Стиль"** в меню **"Редактор"** для применения стиля к выбранным элементам.

## ▣ **Отслеживание изменений**

Использование стилей, и наличие стиля "Удаленный" позволяет автоматически отслеживать изменения в существующей диаграмме. Используйте команду **"Отметить изменения"** в меню **"Редактор/Стиль"**, чтобы позволить или запретить отслеживание изменений.

Когда установлена опция **"Отметить изменения"**, все измененные или добавленные в диаграмму элементы автоматически устанавливаются в стиль "Модифицированный". Когда элемент удаляется, с использованием команд "Удалить" или "Вырезать", они не удаляются из диаграммы визуально, а просто помечаются стилем "Удаленный". Это позволяет пользователю следить за всеми изменениями в диаграмме.

Используйте **"Редактор/Стиль/Устранить все удаленные символы"**, чтобы на самом деле удалить все элементы, помеченные стилем "Удаленный" из диаграммы LD/FBD. Эта команда не принимает во внимание текущий выбор и всегда применяется ко всей диаграмме.

Чтобы восстановить элемент, помеченный стилем **"Удаленный"**, выберите желаемый элемент и примените к нему стиль **"Нормальный"**, стиль **"Модифицированный"** или **"Адаптированный"**. Такая операция может привести к неправильным соединениям (более одной связи подходит к одной и той же точке входа), Это будет определено во время следующей проверки программы.

## A.8 Использование текстового редактора

Этот раздел описывает возможности и команды текстового редактора ISaGRAF, особенно команды, использующиеся для ввода исходных текстов программ ST и IL.

### A.8.1 Команды редактирования

Команды меню **«Редактор»** используются для редактирования текста. Большинство команд оперируют с выделенными символами диаграммы или с символом на текущей позиции курсора.



#### **Удаление и вставка**

Клавиша DEL может быть использована для удаления выделенного текста. Используйте команду **«Редактор / Отменить»** для восстановления элементов, удалённых командой DEL. Команды **«Вырезать»**, **«Копировать»**, **«Вставить»** меню **«Редактор»** используются для перемещения и копирования выделенных элементов текста или для вставки из буфера.

#### **Поиск и замена**

Команды меню **«Редактор / Заменить»**, **«Редактор / Заменить»** используются для нахождения и замены текста в диаграмме. Могут быть найдены любые символьные строки. Поиск может быть осуществлён вверх или вниз от выбранной позиции. Он заканчивается по достижении конца диаграммы.

#### **Переход к строке**

Команда **«Редактор / Перейти на строку»** используется для перемещения курсора на строку с указанным номером. Это может быть полезно для редактирования строки, содержащей ошибку, на которую сослался компилятор программ на ST или IL.



#### **Вставка символа из словаря**

Используйте команду **«Редактор / Вставить переменную»** для вставки на позицию курсора символа переменной или объекта, объявленного в словаре проекта. Символ выбирается из окна диалога, описанного в разделе **«Дополнительно о редакторах программ»** этого документа.

#### **Вставка файла**

Команда **«Редактор / Вставить файл»** вставляет содержимое всего файла на текущую позицию курсора. Следует иметь в виду, что только файлы в ASCII могут быть вставлены этой командой.

## A.8.2 Опции

Команды меню **«Опции»** используются для того, чтобы показать или убрать панель редактирования и выбора шрифта. Выбранный шрифт будет использоваться для любого текста, редактируемого в Инструментарии ISaGRAF. При использовании редактора для ввода программы на ST или IL, команда **«Опции / Показать ключевые слова»** служит для того, чтобы показать или убрать окно, содержащее наиболее употребляемые ключевые слова этих языков.

## A.9 Дополнительно о программных редакторах

Этот раздел содержит полезную информацию о возможностях редактирования, общих для всех программных редакторов ISaGRAF. Это главным образом касается связи различных инструментов ISaGRAF и окон диалога.

### A.9.1 Вызов других инструментов ISaGRAF



#### **Проверка ( компиляция ) программ**

Команда **«Файл / Проверить»** запускает генератор кода ISaGRAF для проверки синтаксиса редактируемой программы. Если она написана на языке SFC, то проверяются как 1-уровень, так и 2-уровень. Когда проверка синтаксиса завершена, окно генерации необходимо закрыть для продолжения редактирования. Если приложение состоит только из одной программы ( редактируемой в данный момент ), то, при отсутствии ошибок, будет создан код приложения. Команда **«Создать / Опции компилятора»** используется для установки параметров компиляции и оптимизации. Обратитесь к разделу **«Использование Генератора Кода»** этого раздела за более подробной информацией о генерации кода и компиляции.



#### **Симуляция или отладка приложения**

Команды **«Файл / Симулировать»** и **«Файл / Отладка»** запускают графический отладчик ISaGRAF как для режима Симуляции, так и для реального соединения. Они перекрывают программу на SFC в режиме отладки. В режиме отладки в программе нельзя произвести никаких изменений.



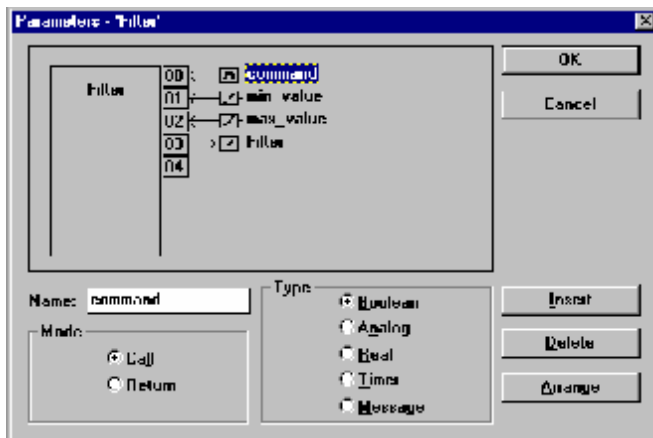
#### **Редактирование словаря переменных**

Команда **«Файл / Словарь»** используется для редактирования словаря переменных текущей программы или текущего приложения. Таким же способом можно редактировать макросы. Описатель **локальный** и макросы относятся к текущей программе.

### A.9.2 Параметры программы

Если редактируемая программа является функцией, функциональным блоком или подпрограммой, то команда **«Файл / Параметры»** используется для определения её параметров вызова и возвращаемых параметров. Эта команда не действует, если редактируемая программа является SFC программой верхнего уровня, от раздела **«Begin»** до **«End»**.

Функции, функциональные блоки или подпрограммы могут иметь до **32** параметров ( ввода или вывода ). Функция или подпрограмма имеет ровно один возвращаемый параметр, имя которого должно совпадать с именем функции, в соответствии с правилами преобразования ST. Следующее окно диалога используется для описания параметров подпрограммы :



Список в верхней левой части окна показывает параметры, в порядке способов вызова : сначала параметры вызова, затем - возврата. Нижняя часть окна показывает подробное описание параметра, выделенного в списке. В качестве параметра может быть использован любой тип данных ISaGRAF. Возвращаемый параметр должен располагаться в списке после параметров вызова. Имена параметров должны подчиняться следующим правилам :

- Длина имени не должна превышать **8** символов
- Первый символ должен быть **буквой**
- Следующие символы могут быть **буквой, цифрой** или символом подчёркивания
- В имени проекта прописной и строчный регистры не различаются

Команда «**Вставить**» используется для вставки нового параметра перед выбранным. Команда «**Удалить**» используется для удаления выделенного параметра. Команда «**Упорядочить**» автоматически сортирует параметры так, что возвращаемый параметр становится последним в списке.

### A.9.3 Остальные команды меню «Файл»

Следующие команды доступны из меню «**Файл**» любого программного редактора.



#### **Открытие другой программы**

Команда «**Файл / Открыть**» позволяет закрыть редактируемую программу и начать редактирование другой программы того же проекта на том же языке. Новая программа заменяет текущую в окне редактирования.



#### **Печать программ**

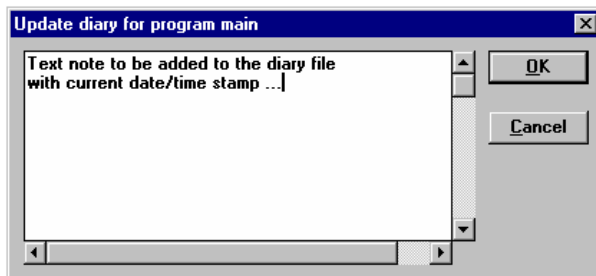
Команда «**Файл / Печать**» выводит редактируемую программу на принтер. Эта команда создаёт эскиз листинга программы. Более подробная информация по этому вопросу получается при использовании генератора документов проекта.

Для графических программ ( SFC, FBD и Quick LD ) Вы можете воспользоваться командой **«Редактор / Копировать рисунок (метафайл)»** для копирования в буфер изображения диаграммы в формате метафайла. Оно может быть вставлено в другое приложение как в текстовом процессоре. Для программ SFC только 1-уровень информация ( диаграммы, нумерация и комментарии 1-уровня ) может быть помещена в метафайл.

#### A.9.4 Обновление дневника программы

Файл дневника, присоединённый к редактируемой программе, может быть введён вручную при помощи команды **«Файл / Дневник»**. Файл дневника обновляется автоматически при компиляции. Вывод компилятора снабжается временной отметкой / отметкой даты.

- ⇒ Если в меню **«Опции»** выбран режим **«Обновить дневник»**, то при каждом сохранении программы на диск появляется следующее окно диалога :

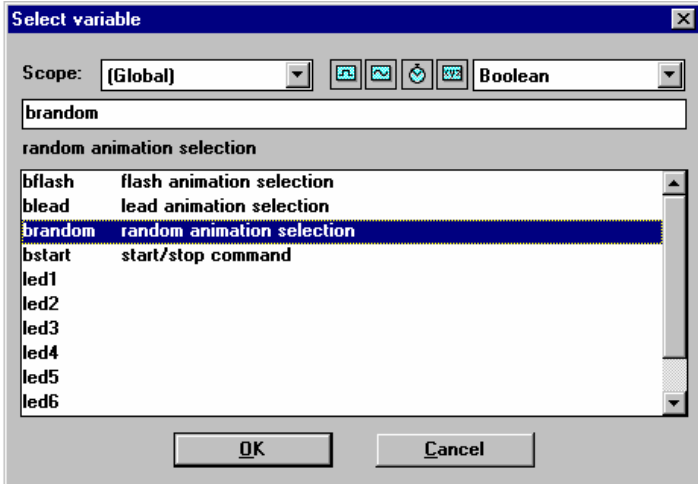


При нажатии кнопки Принять введённый текст сохраняется в конце словаря с отметкой о дате и времени. Эта возможность очень полезна для эксплуатации завершённых программ, так как оно обеспечивает доступ к информации о программе.





#### A.9.5 Выбор переменной из словаря



При редактировании текстовой программы ( IL или ST ) **«Редактор / Вставить переменную»** позволяет выбрать объявленную ранее переменную для вставки на текущую позицию курсора. При редактировании программ FBD или LD требуется для описания контактов, витков, параметров блоков ввода/вывода или переменных FBD. В обоих случаях появляется следующее окно диалога :



Окно выбора «Область» для выбора глобальных или локальных переменных. Окно выбора справа позволяет выбрать тип данных. Маленькие иконки рядом с окном выбора типа - это кнопки, которые используются для выбора наиболее часто используемых типов данных :

-  ..... Boolean
-  ..... Integer / Real
-  ..... Timer
-  ..... Message

Для выбора переменной щёлкните мышью на имени из списка. В результате этого её имя и описание появятся вверху списка. Затем нажмите кнопку «Принять» для подтверждения выбора. Можно также непосредственно ввести имя переменной, не используя список.

## A.9.6 Окно вывода

Следующие команды имеются в меню Инструменты всех языковых редакторов. Они используются для вывода информации в маленькие текстовые списки внизу окна редактирования, и используют ее для просмотра программы.

- "Показать вывод компилятора " показывает в окне вывода сообщение об ошибке последней компиляции редактируемой программы.
- "Найти в диаграмме" находит вхождения текста в редактируемой программе, и представляет их в окне вывода. Для языков SFC and FC, эта команда ищет по всем программам уровня 2.



**"Скрыть список вывода "**                      закрывает окно вывода

Если сообщение об ошибке или вхождении появляются в окне вывода, то двойной щелчок в строке передвигает выделение на соответствующую позицию. Для языков SFC и FC, эта команда открывает соответствующее уровню 2 программное окно.


## A.10 Использование редактора словаря


Словарь ISaGRAF - это средство для редактирования внутренних переменных, переменных ввода/вывода, функциональных блоков и макросов приложения. Словарь группирует все объявления переменных и функциональных блоков приложения и макросов в текстовой форме.

Переменные, функциональные блоки и макросы должны быть объявлены в словаре перед использованием. Переменные и макросы могут использоваться в любом автоматическом языке : SFC, FBD, LD, ST, и IL. Функциональные блоки, используемые в FBD, объявлять не обязательно, так как редакторы FBD и Quick LD автоматически объявляют используемые блоки.


### **Переменные**

Переменные сортируются в соответствии со своей **областью действия** и **типом**. Только переменные с одинаковыми областями действия и типами могут быть введены в одну решётку ввода. Вот основные типы области видимости переменных :


 **ГЛОБАЛЬНАЯ** ..... может быть использована любой программой текущего проекта

 **ЛОКАЛЬНАЯ** ..... может быть использована только одной программой

Ниже приведены основные типы переменных :

 **БУЛЕВСКАЯ** ..... true/false двоичные величины

 **АНАЛОГ** ..... действительные или целые величины

 **ТАЙМЕР** ..... временные величины

 **СООБЩЕНИЕ** ..... символьные строки

Переменная идентифицируется именем, комментарием, атрибутами, сетевым адресом и другими специальными полями. Ниже приведены основные атрибуты переменных :

**ВНУТРЕННЯЯ** ..... переменная в памяти (внутренняя)

**ВХОД** ..... переменная, связанная с устройством ввода

**ВЫХОД** ..... переменная, связанная с устройством вывода




**КОНСТАНТА** ..... внутренняя переменная только для чтения (с начальным значением)

Замечание: **Таймеры** всегда являются **внутренними** переменными. Переменные **ввода** и **вывода** всегда имеют **глобальную** область видимости.

### **Макросы**

Макросы - это синонимы, которые могут быть использованы в любом языке для замены текстовых строк. Заменяемый текст может быть именем переменной,

константой или сложным выражением. Макросы различаются в соответствии с их областью видимости. Только макросы одного типа и области действия могут быть вставлены в одинаковую сетку редактирования. Вот основные виды области действия макросов :

-  **ОБЩАЯ** ..... может быть использована любой программой любого проекта
-  **ГЛОБАЛЬНАЯ** ..... может быть использована любой программой текущего проекта
-  **ЛОКАЛЬНАЯ** ..... может быть использована только одной программой

Макросы идентифицируются именем, блоком описания на ST и комментарием.



### Экземпляры функциональных блоков

Функциональные блоки, используемые в языках ST и IL, должны быть объявлены в словаре. Так как функциональный блок содержит внутренние спрятанные данные, то каждая копия блока должна быть идентифицирована. Следующий пример показывает функциональный блок «R\_TRIG» ( определение поднятой грани ), определённый в библиотеке, который используется для определения фронта различных переменных. Каждая копия блока должна быть идентифицирована уникальным именем. Определение типа блока и его параметров производится при помощи менеджера библиотек :

**Имя блока:** R\_TRIG  
**Параметры:** Input=CLK  
Output=Q

При помощи редактора словаря образуется имя :

**Имя экземпляра:** TRIG\_B1                      **Имя блока:** R\_TRIG  
**Имя экземпляра:** TRIG\_B2                      **Имя блока:** R\_TRIG

Объявленные блоки могут быть использованы в ST программе :

```
TRIG_B1 (b1);
edge_b1 := TRIG_B1.Q;    (* b1 определение фронта переменной *)
TRIG_B2 (b2);
edge_b2 := TRIG_B2.Q;    (* b2 определение фронта переменной *)
```

Объявленные экземпляры функциональных блоков могут быть **ГЛОБАЛЬНЫМИ** ( доступные в любой программе проекта ) или **ЛОКАЛЬНЫМИ**, для одной программы. Функциональные блоки, используемые в языках FBD и LD, объявлять не нужно, так как это делает редактор FBD.



(\* функциональные блоки всегда имеют то же имя, что и блок, объявленный в библиотеке. Редактор FBD автоматически объявляет реализацию каждый раз при вставке блока из библиотеки \*)

Экземпляры функциональных блоков, автоматически объявленные редакторами FBD и LD, всегда локальны для редактируемой программы.

## ▬ Сетевые адреса

Сетевой адрес является **необязательным**. Переменная с ненулевым сетевым адресом может быть **прослежена** внешней системой ( например, процессом визуализации системы ) в процессе выполнения. В более общем случае, сетевой адрес обеспечивает механизм идентификации для каждой работающей коммуникационной системы, которая не поддерживает символьное имя. Сетевой адрес может быть определён для каждой переменной, в том числе при её создании или изменении.

### А.10.1 Главное окно словаря

✚ Окно редактирования словаря показывает список переменных с одинаковым типом и областью действия. Тип и область действия всегда указаны в заголовке.

✚ Окно редактирования показывает только основные поля описания переменных : имя, атрибуты, сетевой адрес и комментарий. Полное описание выделенных переменных всегда дано в окне текущего состояния. Используйте следующие кнопки на инструментальной панели для выбора области действия редактируемых переменных :



**ОБЩАЯ** ..... может быть использована любой программой любого проекта



**ГЛОБАЛЬНАЯ** ..... может быть использована любой программой текущего проекта



**ЛОКАЛЬНАЯ** ..... может быть использована только одной программой

Используйте "Tab" управление в полосе заголовка, для выбора типа объекта редактирования:

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



Используйте поле ввода текста слева от панели для поиска префикса имени переменной. В данном случае поиск производится по всему списку от начала. Кроме этого, для поиска доступны команды меню «**Найти**». Поиск не различает **регистров**. Поиск может проводится **вперёд** или **назад** по отношению к выделенной переменной.

## A.10.2 Управление переменными

Доступные команды меню **«Файлы»** работают с выделенным классом переменных, реализаций функциональных блоков и макросов. Используйте команду **«Другой»** для выбора типа и области действия редактируемых переменных.



### **Печать переменных**

Используйте команду **«Файл / Печать»** для того, чтобы распечатать список редактируемых объектов на стандартном принтере Windows. Распечатка включает в себя полное описание каждого объекта текущей редактируемой группы.



### **Создание новых переменных**

Команда **«Файл / Новый»** позволяет пользователю создавать новые переменные, экземпляры функциональных блоков и макросы текущего типа и области действия. Новая переменная вставляется сразу же после текущей. При запуске этой команды открывается окно диалога, в которое требуется ввести описание новой переменной. После этого нажатие кнопки **«Сохранить»** помещает созданную переменную в список. Окно ввода открывается снова, так что пользователь может ввести описание новой переменной. Нажатие кнопки **«Отказ»** останавливает процесс создания переменной.



### **Изменение существующей переменной**

Команда **«Редактор»** меню **«Редактор»** позволяет пользователю изменить описание текущей переменной. При запуске команды открывается окно ввода для изменения описания текущей переменной. При нажатии кнопки **«Сохранить»** изменения сохраняются. Кроме того, можно нажать кнопки **«Следующий»** или **«Предыдущий»** для распространения изменений на другие переменные. Нажатие кнопки **«Отказ»** останавливает процесс создания переменной.



### **Удаление и вставка**

Инструменты редактирования словаря ISaGRAF содержат выделение **нескольких линий**. Многие команды могут оперировать со списком выделенных переменных. Ниже приведены доступные команды меню **«Редактор»** :

- КОПИРОВАТЬ ...** Копировать выделенную группу переменных в буфер словаря
- ВЫРЕЗАТЬ .....** Копировать выделенную группу переменных в буфер словаря и удалить её из списка
- УДАЛИТЬ .....** Удалить выбранную группу переменных из редактируемого списка
- ВСТАВИТЬ .....** Вставить буфер словаря перед выделенной переменной

Функции Вырезать/Копировать/Вставить могут быть применены к различным спискам переменных с одинаковыми параметрами.



### **Сортировка переменных**

Команда «**Инструменты / Сортировка**» сортирует переменные или макросы текущего списка редактирования. Порядок сортировки определяется атрибутами переменных :

- сначала внутренние переменные
- затем переменные ввода
- наконец, переменные вывода

Переменные с одинаковыми атрибутами сортируются в алфавитном порядке. Макросы всегда сортируются в алфавитном порядке.

## **Установка сетевых адресов**

Сетевой адрес не является обязательным. Переменная с ненулевым сетевым адресом может быть прослежена внешней системой ( например, процессом визуализации системы ) в процессе выполнения. Сетевой адрес может быть определён для каждой переменной, в том числе при её создании или изменении. Команда «**Редактор / Перенумеровать**» позволяет пользователю установить сетевой адрес для текущей группы переменных. Ввод шестнадцатеричного **базового адреса** ( адреса первой переменной группы ) приводит к тому, что для переменных группы устанавливаются **последовательные сетевые адреса**. При вводе нулевого базового адреса всех выделенных переменных сбрасываются на ноль.



## **Импорт булевых строк “true/false”**

При редактировании макросов команда «**Инструменты / Импорт определений true/false** » позволяет пользователю автоматически определить строки, присоединённые к булевым переменным, в качестве ключевых слов языка. Они должны быть объявлены как макросы. Эта команда ищет булевы истинностные строки в объявлениях той же области, что и редактируемый, для изменения макросов.

### **A.10.3 Описание объектов**

Для любой переменной, функциональных блоков или макросов должно существовать полное описание. Поля описания различаются в зависимости от типа объекта. Следующие поля определены для любого типа :

- Имя** .....Имя переменной; первый символ должен быть буквой, остальные - буквы, цифры или символы подчёркивания.
- Сетевой адрес** .....Шестнадцатеричный сетевой адрес ( не является необходимым ). Переменная с ненулевым сетевым адресом может быть прослежена внешней системой.
- Коммент**.....комментарий описания переменной
- Хранить**.....эта опция показывает, что переменная должна сохраняться в резервной памяти



Это описание полей для **булевых** переменных :

- Атрибут** .....определяет, является переменная внутренней, константой, ввода/вывода

**"False" строка** ..... строка, используемая в качестве ложного значения во время отладки  
**"True" строка** ..... строка, используемая в качестве истинного значения во время отладки  
**Нач. знач. true** ..... если эта опция установлена, то начальное значение истинно, иначе - ложно



Это описание полей для **целых или вещественных** аналоговых переменных :

**Атрибут** ..... определяет, является переменная внутренней, константой, ввода/вывода  
**Формат** ..... определяет целую или вещественную переменную. Формат может быть выбран во время отладки.  
**Единицы** ..... строка, используемая для определения физического цикла отладки  
**Преобраз** ..... Имя таблицы или функции преобразования, присоединённой к переменной ( только для ввода/вывода )  
**Нач. знач.** ..... Начальное значение переменной. По умолчанию 0.



Это описание полей для переменных **времени** :

**Атрибут** ..... определяет, является переменная внутренней, константой  
**Нач. знач.** ..... Начальное значение переменной. По умолчанию 0.



Это описание полей для переменных типа **сообщение** :

**Атрибут** ..... определяет, является переменная внутренней, константой, ввода/вывода  
**Макс. длина** ..... максимальное количество символов, которое может храниться в строке сообщения  
**Нач. знач.** ..... Начальное значение переменной. Его длина не может превышать максимального количества символов. По умолчанию пустая строка.



Это описание полей для макросов :

**Имя** ..... Имя, используемое в ST; первый символ должен быть буквой, остальные - буквы, цифры или символы подчёркивания.  
**Определ** ..... Строка, подчиняющаяся правилам синтаксиса ST, которая заменяется макросом при компиляции. Пример : Name = PI-Equivalence = 3.1459  
**Коммент** ..... Произвольный Комментарий для макроса



Это описание остальных полей для реализаций функциональных блоков :

**Имя** .....Имя экземпляра, используемое в ST; первый символ должен быть буквой, остальные - буквы, цифры или символы подчёркивания.

**Тип** ..... имя соответствующего функционального блока в библиотеке

**Коммент** ..... Произвольный Комментарий для функционального блока

#### A.10.4 Быстрое объявление

Команда "**Инструменты / Быстрое объявление** " позволяет вам объявить несколько переменных одновременно. Переменным, созданным с помощью быстрого объявления, присваиваются имена в соответствии с числовым соглашением. Для этого, вы должны определить:

- индекс (число) первой и последней переменной,
- текст, который должен быть добавлен перед или после числа в символе переменной

- количество цифр, используемое для выражения числа в символах переменных.

Дополнительно, вы можете определить основные атрибуты созданных переменных (внутренняя, вход или выход...), плюс некоторые свойства, зависящие от типа переменной (атрибут "Хранить", целый или действительный формат, максимальная длина строки сообщения).

Вы всегда должны определять текст, который нужно вводить перед номером переменной так, как символ переменной не может начинаться с цифры. Если "количество переменных" установлено в "Auto", ISaGRAF форматирует количество переменных к минимальному необходимому количеству цифр. Когда определено количество цифр, ISaGRAF форматирует все числа, к определенной длине добавляя спереди символы '0'. Установка фиксированного количества цифр для номеров переменных может быть полезна для предотвращения неправильной лексикографической сортировки. Ниже дан пример.

Пример: Это установка для быстрого объявления:

<b>Numbering:</b>			
<b>From:</b>	<input type="text" value="9"/>	<b>To:</b>	<input type="text" value="11"/>
<b>Digits:</b>	<input type="text" value="auto"/>		
<b>Symbol:</b>			
<b>Name:</b>	<input type="text" value="Var"/>	<b>##</b>	<input type="text" value="xx"/>

будут созданы три следующие переменные:

**Var9xx    Var10xx    Var11xx**

Пример: Это установка для быстрого объявления:



<b>Numbering:</b>		
<b>From:</b>	<input type="text" value="1"/>	<b>To:</b> <input type="text" value="100"/>
<b>Digits:</b>	<input type="text" value="3"/>	
<b>Symbol:</b>		
<b>Name:</b>	<input type="text" value="MyVar"/>	<b>##</b> <input type="text"/>

будут созданы 100 переменных с именами от **MyVar001** до **MyVar100**

### A.10.5 Адресная карта Modbus для SCADA

"Сетевые адреса" ISaGRAF часто используются для установления связи между системой ISaGRAF и SCADA на основе Modbus. В этом случае, SCADA является мастером Modbus, а целевая задача ISaGRAF действует как подчиненный Modbus. Сетевые адреса используются для создания виртуальной карты Modbus для всех переменных ISaGRAF, которые должны управляться из SCADA. **"Инструменты / Карта адресов Modbus для SCADA"** создает виртуальную карту Modbus с переменными приложения.

Средства карты показывают два списка. Верхний список – это сегмент (положения 4096) карты Modbus, показывающий нанесенные на карту переменные (имеющие сетевые адреса). Нижний список показывает не нанесенные на карту переменные (без определенных сетевых адресов). Адрес "0" не может использоваться в карте для переменной.

Используйте команды **"Поместить на карту выбранную переменную"** и **"Удалить переменную из карты"** меню **"Редактор"**, чтобы перемещать переменные из одного списка в другой, и строить, таким образом, карту. Такие же действия могут быть выполнены путем двойного щелчка на символе переменной в списке, чтобы послать ее в другой список. В любой момент вы можете использовать список "Сегмент", чтобы посмотреть другой сегмент в карте.

Команды меню **"Опции"** могут быть использованы для того, чтобы показать адреса в десятичном или шестнадцатеричном виде.

Команда **"Редактор / Заменить"** используется для поиска объявленной переменной, нанесена она на карту или нет.

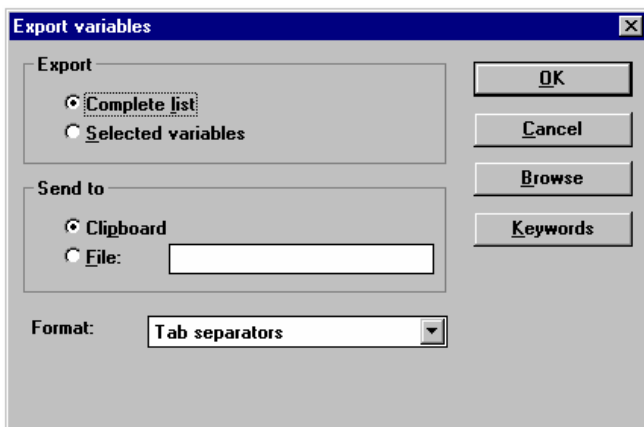
### A.10.6 Обмен информацией с другим приложением

Редактор словаря ISaGRAF поддерживает функции импорта/экспорта для обмена информацией с другим приложением, таким как текстовый процессор, таблицы или база данных. Эти команды сгруппированы в меню **«Редактор»**. Команда **«Экспортировать текст»** генерирует текстовое ASCII описание полей, описывающих группу редактируемых объектов и сохраняет это описание в Буфере Обмена Windows или в файле. Такая информация может быть использована в другом приложении. Команда **«Импортировать текст»** импортирует поля описания объявления переменной в формате ASCII, которые находятся в Буфере Обмена Windows или в файле. Эта команда обновляет список содержимым

импортируемых полей. Как правило, это информация, созданная другим приложением.

## Экспорт данных

Следующее окно диалога появляется при запуске команды «**Экспортировать текст**». Оно позволяет пользователю управлять экспортированием :



«**Полный список**» экспортируется весь редактируемый список. Выделенные элементы игнорируются. При выборе «**Выбранные переменные**» экспортируются только выделенные переменные.

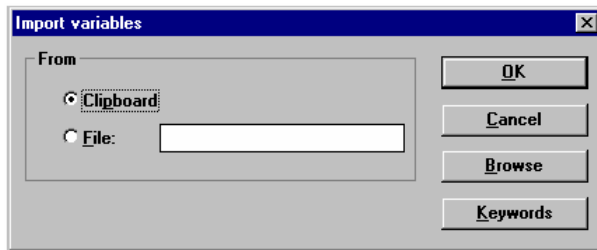
При установке опции «**Клипборд**» экспортируемая информация сохраняется в формате ASCII в Буфере Обмена Windows. В дальнейшем к этому тексту можно применить команду «Вставить». При установке опции «**Файл**» экспортируемый текст сохраняется в ASCII файле. Для определения имени этого файла используется команда «**Смотреть**».

После этого пользователь выбирает формат экспортируемого текста. Возможные форматы описаны далее. Нажатие кнопки «**Принять**» запускает экспортирование, кнопки «**Отказ**» отменяет его.

Все поля выделенных объектов экспортируются в виде текста в стандартном порядке. Первая строка экспортируемого текста содержит имя поля . Каждый объект описывается на одной строке текста. Конец строки - в стандартном формате MS-DOS «**0d-0a**». Идентификаторы полей в первой экспортируемой строке могут быть изменены при помощи кнопки «**Ключевые слова**». Эта команда описана далее.

## Импорт данных

Следующее окно диалога появляется при запуске команды «**Импорт текст**». Оно позволяет пользователю управлять импортированием :



При выборе опции "Клипборд" импортируемая информация берётся из буфера обмена в формате . При выборе опции "Файл" информация берётся из файла. В этом случае нужно ввести имя файла, для чего может быть использована команда "Смотреть" .

Импортирующая функция автоматически распознаёт формат, используемый в тексте. Возможные форматы описаны в последующих разделах. Нажатие кнопки «Принять» запускает экспортирование, кнопки «Отказ» отменяет его. Идентификаторы полей в первой экспортируемой строке могут быть изменены при помощи кнопки «Ключевые слова». Эта команда описана далее.

Первая строка текста должна содержать имя поля в соответствии с порядком использования следующих линий. Каждый объект описывается на одной строке текста. Конец строки - в стандартном формате MS-DOS «0d-0a». Поля могут следовать в любом порядке. Если некоторые поля опущены, то они автоматически заполняются значениями по умолчанию. Если импортируемый объект уже существует в списке, то пользователь должен подтвердить необходимость замены его на импортируемый. Описание объектов обновляется импортированными значениями.

## ☰ **Возможные текстовые форматы**

Ниже приведён список возможных текстовых форматов для экспортирования. Команда импортирования автоматически распознаёт эти форматы :

- tab separators ( разделение табуляцией )

Описание : поля разделены символом табуляции

*Пример:*

Name	Attribute	Comment
level	Внутренняя	internal calculated water level
alm1	output	main
		выход основной тревоги

- comma separator ( разделение запятой )

Описание : поля разделены запятой

*Пример:*

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

- semicolon separators (разделение точкой с запятой)

Описание : поля разделены точкой с запятой

Пример: Name;Attribute;Comment  
level;Внутренняя;Внутренняя calculated water level  
alm1;output;main alarm output

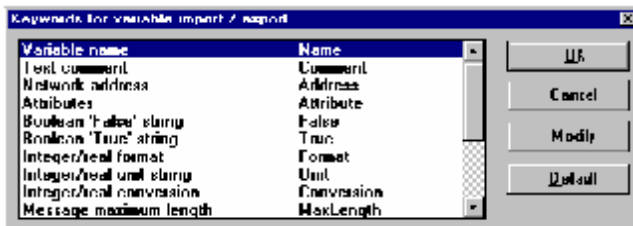
- commas commas and quotes (запятые и кавычки)

Описание : поля разделены запятыми.  
Каждое поле взято в кавычки.

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

## Ключевые слова

Имена, используемые для идентификации полей в первой строке импортируемого или экспортируемого текста можно изменить нажатием кнопки «Ключевые слова». Эта команда открывает следующее окно диалога :



В окне показан список полей объектов и соответствующие им ключевые слова. Для изменения ключевого слова пользователь должен выбрать поле в списке и нажать кнопку «Изменить». Нажатие кнопки «Умолчания» восстанавливает изначальные значения списка ключевых слов. Синтаксис ключевых слов должен подчиняться следующим правилам :

- имени не должна превышать **16** символов
- символ должен быть **буквой**
- символы могут быть **буквой, цифрой** или символом подчёркивания
- одинаковые имена не могут быть использованы для разных ключевых слов

Ниже приведены стандартные ключевые слова ISaGRAF :

Имя объекта .....	<b>Name</b>
Комментарий .....	<b>Comment</b>
Сетевой адрес .....	<b>Address</b>
Attributes (Внутренняя, input, output).....	<b>Attribute</b>
БУЛЕВСКАЯ 'False' string .....	<b>False</b>
БУЛЕВСКАЯ 'True' string .....	<b>True</b>
Analog format (real or integer).....	<b>Format</b>

Analog unit string .....	<b>Unit</b>
Analog conversion name.....	<b>Conversion</b>
Message maximum length .....	<b>MaxLength</b>
Function block library type.....	<b>Library</b>
Defined word equivalence .....	<b>Equivalence</b>
Внутренняя attribute .....	<b>Внутренняя</b>
Input attribute.....	<b>Input</b>
Output attribute .....	<b>Output</b>
Constant attribute.....	<b>Constant</b>
Real analog format .....	<b>Real</b>
Integer analog format .....	<b>Integer</b>








## A.11 Использование редактора соединений В/В.

Цель операции соединения В/В - это установление логической связи между переменными В/В приложения и физическими каналами плат, существующими на целевой машине. Для создания этой связи пользователь должен определить и установить все платы целевой машины, и поместить переменные В/В на соответствующие каналы В/В.




Список слева показывает шасси целевой машины с **разъемами плат**. Разъем может быть пуст, или использован одной платой В/В или более сложным оборудованием. Каждый разъем определяется **порядковым номером**. Шасси может содержать до **255** плат. Список справа показывает параметры плат и переменные, подключенные к выбранной плате. Плата может иметь до **128** каналов В/В.

### Иконки




Иконки, отображенные на передней грани, показывают тип и атрибуты переменных, которые могут быть подключены к каналам платы. Система ISaGRAF не позволяет подключение переменных разного типа на той же самой плате. Ниже значение используемых иконок:

-  ..... тип логическое
-  ..... тип целое/вещественное
-  ..... тип сообщение
-  ..... входы - нет подключенных каналов
-  ..... выходы - нет подключенных каналов
-  ..... входы - минимум один канал подключен
-  ..... выходы - минимум один канал подключен

Ниже показаны иконки, используемые для отображения типа устройств В/В, установленного в разъем:

-  ..... комплексное оборудование В/В
-  ..... действительное оборудование В/В
-  ..... виртуальное оборудование В/В

Ниже показаны иконки, используемые для отображения параметра или канала:

-  ..... параметр платы
-  ..... свободный канал
-  ..... занятый канал



### Передвижение плат в списке

Используйте эти кнопки в панели инструментов или **“Редактор / Передвинуть вверх/вниз”** в меню команд для передвижения выбранной платы В/В на одну линию вверх или вниз в главном списке. Команда **“Редактор / Вставить слот”** устанавливает пустой разъем в текущей позиции.

## A.11.1 Определение плат В/В

Меню “**Редактор**” содержит команды для определения выбранной платы (устанавливает ее параметры) и подключения переменных В/В к их каналам.



### **Выбор типа плат В/В**

Перед подключением переменных В/В к плате, должна быть введена идентификация платы. Библиотека предопределенных плат доступна на ISaGRAF workbench. Эта библиотека может быть скомпилирована одним или более поставщиками устройств В/В. Команда “**Редактор / Установить плату/ оборудование**” используется для установки идентификации платы. Эта команда может быть использована для выбора либо одиночной платы, либо сложного оборудования В/В из библиотеки ISaGRAF. Также возможна установка соответствующей платы или оборудования двойным щелчком мыши на разъеме. Все каналы одной платы имеют один и тот же тип (логическое, целое/вещественное или сообщение) и направление (вход или выход). Переменные типа вещественное и целое не различаются в процессе подключения В/В. Сложное оборудование В/В представляет устройство В/В с каналами различных типов и направлений. Сложное оборудование В/В представляется как список одиночных плат В/В. Оно использует только один разъем в списке шасси.



### **Удаление платы**

Команда “**Редактор / Очистить слот**” используется для удаления выбранной платы В/В. Если переменные уже подключены к соответствующим каналам, то они автоматически отключаются при освобождении разъема.



### **Действительные платы и виртуальные платы**

Команда “**Редактор / Действительная /виртуальная плата**” устанавливает действительность выбранной платы или сложного оборудования. Следующие иконки используются в списке шасси для отображения действительности платы:



.....реальная плата



.....виртуальная плата

В **действительном режиме** переменные В/В напрямую соединены к соответствующим устройствам В/В. Операции ввода или вывода в программах приложений напрямую связаны с состояниями соответствующих входов или выходов существующих устройств В/В. В **виртуальном режиме** переменные В/В обрабатываются именно как внутренние переменные. Они могут быть прочитаны или изменены при помощи отладчика, так что пользователь может имитировать обработку В/В, но без соединения с внешним миром.



### **Технические замечания**

Команда “**Инструменты / Техническое замечание**” отображает пользовательскую подсказку выбранной платы или сложного оборудования В/В. Техническое замечание написано поставщиком платы. Оно содержит всю информацию об управлении платой. Оно также описывает значение параметров платы.



## **Удаление присоединенных переменных**

Команда "Инструменты / Освободить каналы платы" отсоединяет все переменные В/В уже соединенные на выбранной плате.




## **Определение комментариев для свободных каналов**

Текст комментариев объявленной переменной В/В отображается с ее именем в списке платы. Так как ISaGRAF также позволяет использовать непосредственно представленные переменные (в процентной нотации), то комментарии также могут быть введены для свободных каналов. Используйте команду **«Инструменты / Установить комментарий канала»** для ввода комментариев для текущего свободного канала выбранного в списке платы. Эта команда не может быть использована для смены комментария переменной В/В заданного в словаре проекта.

### **А.11.2 Установка параметров плат**



Для установки значения параметра платы, пользователь должен дважды щелкнуть мышкой на его имени в списке справа. Также возможно выбрать команду **«Установить канал /параметр»** из меню **«Редактор»**. Параметры перечислены в начале списка. Следующая иконка используется для представления их в списке:

 .....параметр платы

Значение и входной формат параметра разрабатывается поставщиком соответствующей платы В/В или оборудования. Используйте команду **«Инструменты / Техническое замечание»** или обратитесь к описанию для получения дополнительной информации о параметрах платы.

### **А.11.3 Подключение каналов В/В**

Для установления соединения канала пользователь должен дважды щелкнуть на его расположении в списке справа. Также возможно выбрать и запустить команду **«Редактор / Установить канал/параметр»**. Следующие иконки используются для представления каналов в списке:

-  .....свободный канал
-  .....подключенный канал

Список содержит все переменные, которые подходят к выбранной плате по типу и направлению. Здесь перечислены только те переменные, которые еще не подключены. Кнопка **«Соединить»** подключает переменные, выбранные в списке к выбранному каналу. Кнопка **«Освободить»** удаляет (отключает) переменные с выбранного канала. Кнопки **«Следующий»** и **«Предыдущий»** используются для выбора другого канала платы. Расположение выбранного канала всегда отображается в заголовке окна диалога.



## A.11.4 Непосредственно представленные переменные

Свободные каналы - это каналы, которые не связаны с объявленными переменными В/В. ISaGRAF позволяет использование **непосредственно представленных переменных** в исходном коде программ для представления свободного канала. Идентификатор непосредственно представленной переменной всегда начинается с символа "%".

Ниже даны именные условия непосредственно представленных переменных для канала одиночной платы. "s" - это номер разъема платы. "c" - это номер канала.

%IXs.c .....свободный канал платы ввода типа логическое  
%IDs.c .....свободный канал платы ввода типа целое  
%ISs.c .....свободный канал платы ввода типа сообщение  
%QXs.c .....свободный канал платы вывода типа логическое  
%QDs.c .....свободный канал платы вывода типа целое  
%QSS.c .....свободный канал платы вывода типа сообщение

Ниже даны именные условия непосредственно представленных переменных для канала сложного оборудования. "s" - это номер разъема платы. "b" - это индекс одиночной платы внутри сложного оборудования. "c" - это номер канала.

%IXs.b.c .....свободный канал платы ввода типа логическое  
%IDs.b.c .....свободный канал платы ввода типа целое  
%ISs.b.c .....свободный канал платы ввода типа сообщение  
%QXs.b.c .....свободный канал платы вывода типа логическое  
%QDs.b.c .....свободный канал платы вывода типа целое  
%QSS.b.c .....свободный канал платы вывода типа сообщение

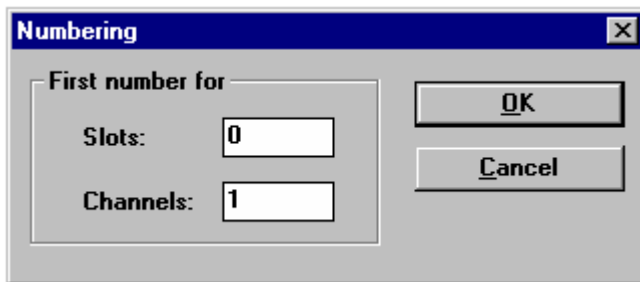
Ниже даны примеры:

%QX1.6      6-й канал платы № 1 (вывод типа логическое)  
%ID2.1.7    7-1 канал платы № 1 оборудования № 2 (ввод типа целое)

А Непосредственно представленные переменные не могут иметь тип **"вещественное"**.

## A.11.5 Нумерация

Используйте команду **"Опции / Нумерация"**, чтобы установить условия нумерации. Вы можете определить номер, используемый для первого слота и номер, используемый для первого канала каждой платы в следующем диалоге:



По умолчанию, нумерация слотов начинается с индекса "0", а нумерация каналов начинается с индекса "1".

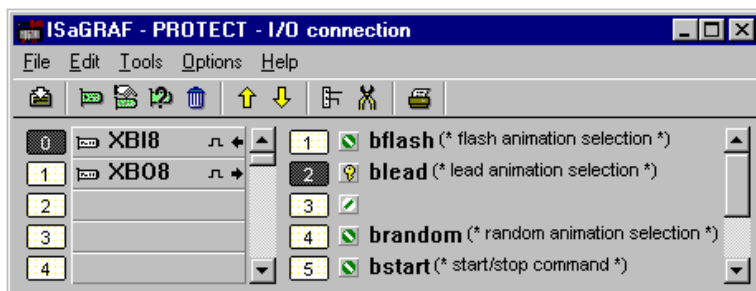
**Предупреждение:** будьте осторожны при смене соглашений о нумерации так, как это оказывает влияние на символы, используемые для непосредственно представленных переменных и может привести к ошибкам компиляции, если непосредственно представленные переменные используются в существующих программах.

#### A.11.6 Установка индивидуальных защит

Система разработки ISaGRAF обеспечивает полную систему защиту данных основанную на иерархических паролях. Соединения В/В могут быть глобально защищены паролем. Дополнительно, ISaGRAF позволяет вам установить индивидуальную защиту на любой канал В/В. Это подразумевает, что:

- пароли всегда определяются в системе определения пароля (используйте команду **"Проект / Установить пароль"** окна Управление Проектами) так, что имеются индивидуальные уровни защиты.
- вы используете для индивидуальной защиты уровни защиты с более высоким приоритетом по сравнению с глобальной защитой В/В.

Когда канал В/В имеет индивидуальную защиту, маленькая иконка нарисована рядом с его именем в окне соединений В/В:



Используйте команды **"Установить защиту"** и **"Уничтожить защиту"** меню **"Редактор"**, чтобы установить или удалить индивидуальную защиту для выбранного канала. Обе команды предлагают вам ввести действительный пароль так, что уровень защиты может быть присоединен к каналу. Затем, каждый раз, когда вы хотите изменить соединение с каналом вы должны ввести пароль достаточного уровня приоритета.

Предупреждение: Если канал защищен паролем, то соответствующий пароль удаляется из системы защиты, и если не определено пароля с более высоким уровнем, соединение с каналом не может больше быть изменено, без нового пароля с достаточным уровнем.

## A.12 Создание таблиц преобразования

Система разработки ISaGRAF позволяет пользователю создавать таблицы преобразования. Таблица преобразования – это набор точек, используемых для аналогового преобразования. Таблица преобразования может быть подключена к входной или выходной аналоговой переменной. Таблица создает пропорциональное соотношение между электрическими значениями (считанных с входных датчиков или посланных на выходное устройство) и физическими значениями (используемыми в программировании приложения).

Таблицы преобразований редактируются с помощью диалога команды **"Инструменты / Таблицы преобразований"** словаря ISaGRAF.

Определенная таблица преобразований может быть использована для фильтрации значений любой входной или выходной аналоговой переменной выбранного проекта. Для подключения таблицы преобразования к переменной, должен быть открыт редактор объявления переменных. Входная или выходная аналоговая переменная после должна быть выбрана и ее параметры отредактированы. Переменная не может быть подключена к таблице преобразования, которая еще не определена.

### A.12.1 Основные команды

Диалог **"Таблицы преобразований"** показывает список определенных таблиц преобразования, и содержит кнопки для основных команд, чтобы редактировать существующую таблицу (определять ее точки), чтобы создавать новую таблицу, и также переименовывать и удалять таблицу. Нажмите Принять, чтобы выйти из диалога "Таблицы преобразований" и сохранить их на диске.



#### **Создание новой таблицы**

Команда "Новый" позволяет пользователю создать новую таблицу преобразования. До 127 таблиц преобразования может быть создано для каждого проекта. Только используемые таблицы (которые подключены к аналоговым переменным) включаются в исполняемый код приложения. Наименование таблицы должно удовлетворять следующим правилам:

- Длина имени не должна превышать **16** символов
- Первый символ должен быть **буквой**
- Следующие символы должны быть **буквами, цифрами** или **символом подчеркивания**
- Верхний и нижний регистры в наименовании таблицы не различаются



#### **Изменение содержания таблицы**

Команда **"Редактор"** используется для ввода точек таблицы, выбранных из списка. Также возможно дважды щелкнуть на имени таблицы или нажать клавишу **ENTER** для вызова команды **"Редактор"**. Команда **"Редактор"** автоматически вызывается когда создается новая таблица. Как минимум две точки должны быть введены для каждой таблицы.

## A.12.2 Ввод точек таблицы

Окно диалога “Редактор” позволяет пользователю определять точки таблицы преобразования. Окно, показывающее на левой стороне список точек, уже определено. Нижнее правое окно показывает определенную таблицу как графическую кривую.

Точки вводятся при помощи команд окна. Пользователь должен подчиняться определенному количеству правил для определения точек, описанных в конце этой главы. Окно слева всегда содержит список существующих точек для текущей редактируемой таблицы. Столбик слева показывает электрические (внешние) значения точки. Столбик справа показывает физические (внутренние) значения. Пользователь должен выбрать точку в списке для того, чтобы изменить ее значение или очистить (удалить) ее. Последний выбор в списке (“... ..”) используется для определения новой точки. Окно внизу справа показывает текущую редактируемую таблицу как графическую кривую. Оси и координаты не показаны, так как это пропорциональное представление кривой. Это представление полезно как быстрая проверка того, что кривая правильно определена.

### ⇒ **Определение новой точки**

Для определения новой точки выберите последнее вхождение (“... ..”) в списке точек. Это также режим по умолчанию, когда запускается определение новой таблицы преобразования. Пользователь должен ввести электрические (внешние) и физические (внутренние) значения каждой точки. Значения сохраняются как числа с плавающей точкой простой точности. Помните, что как минимум **две точки** должны быть введены для определения кривой. Когда оба значения введены, нажатие кнопки “Сохранить” добавляет точку в таблицу. Максимум 32 точки могут быть определены для каждой таблицы преобразования.

### ⇒ **Изменение точки**

Для изменения значения существующей точки сначала выберите ее из списка. После этого новые электрические (внешние) и физические (внутренние) значения могут быть введены. Значения сохраняются как числа с плавающей точкой простой точности. Когда оба значения введены, нажатие кнопки “Сохранить” обновляет точку в таблице.

### ⇒ **Очищение точки**

Существующая точка очищается выбором ее в списке и нажатием кнопки “Сбросить”. Помните, что как минимум **две точки** должны быть введены для определения таблицы.

## A.12.3 Правила и ограничения

Необходимо следовать правилам показанным ниже для определения таблиц преобразования. Таблица может быть использована для преобразования и входных и выходных аналоговых переменных:

- Две точки не могут быть определены с одним и тем же электрическим значением

- Кривая должна непрерывно возрастать или убывать
- Две точки не могут быть определены с одним и тем же физическим значением

Следующие ограничения применяются при определении таблиц преобразования для проекта:

- Не больше чем 127 таблиц преобразования может быть определено в одном проекте
- Не больше чем 32 точки может быть определено для одной таблицы преобразования.

## A.13 Использование генератора кода

Окно генератора кодов автоматически открывается командами “Проверить” и “Создать” из другого окна ISaGRAF Workbench. Окно генератора кодов не закрывается автоматически, когда заканчивается затребованная операция генерации кодов, так что пользователь все еще будет иметь доступ ко всем командам генерации кодов и опциям из меню окна.

### A.13.1 Основные команды

Меню “Файл” содержит команды для проверки синтаксиса программы и генерации кода.

#### ☰ **Генерация кода приложения**

Команда “Создать” конструирует целиком код проекта. Пред генерацией чего-либо эта команда проверяет синтаксис определений и программ. Любая ошибка, которая не может быть обнаружена во время компиляции одиночной программы, обнаруживается в процессе генерации кода. Это применимо к таблицам преобразования, соединениям переменных В/В и связям с библиотеками. Генератор кодов останавливает компиляцию программы, когда обнаруживаются ошибки. Эта программа должна быть исправлена перед тем, как будет продолжена генерация кодов. Программы, которые уже были проверены (и ошибки не обнаружены) и не были изменены со времени выполнения последней операции “Проверить”, не компилируются по новой. Проверка объявления переменных и проверка связности приложения производится всегда. Во время проверки программы операция “Создать” может быть прервана нажатием клавиши **ESCAPE**.

Замечание: Если объявление локальной переменной программы было изменено, то такая программа проверяется. Если глобальная переменная была изменена, то проверяются все программы.

#### ☰ **Проверка синтаксиса программы**

Команда “Проверить программу” позволяет пользователю проверить только одну программу. Выбранная программа компилируется, даже если она не была изменена со времени последней проверки. Команда “Проверить словарь” позволяет пользователю проверить объявления всех переменных проекта. “Проверить все программы” проверяет синтаксис всех программ проекта, даже если некоторые из них не были изменены. Эта команда не останавливается когда обнаруживается ошибка в программе. Это может быть использовано для создания полного списка всех ошибок, оставшихся в программах проекта. Эта команда может быть прервана нажатием клавиши **ESCAPE**.

#### ☰ **Имитация изменения**

Команда “Коснуться” имитирует изменение всех программ проекта, так что они все будут проверены во время следующей операции “Создать”. Команда “Открыть” используется для открытия последней проверенной программы. Эта

команда очень полезна для прямого доступа к программе, в которой была обнаружена синтаксическая ошибка.

## A.13.2 Опции компилятора

Команда “**Опции компилятора**” применяется для установки основных параметров, используемых генератором кодов ISaGRAF для построения и оптимизации получаемых кодов. Цель этой команды - выбрать тип кода, который должен быть сгенерирован, в соответствии с целевой задачей ISaGRAF, и установить параметры оптимизатора, в соответствии с ожидаемым временем компиляции и рабочими требованиями приложения.

Команда “**Выгрузить**” открывает второй диалог с другими опциями, которые позволяют загружать упакованный исходный код для того, чтобы разрешить возможность выгрузки. Смотрите документацию по “Выгрузке”.

### ☰ **Выбор целевой задачи**

Верхний список показывает перечень возможных кодов назначения, которые могут быть сгенерированы. Знак “>” используется для обозначения выбора. Генератор кодов ISaGRAF может генерить до 3 различных кодов в одной операции компилирования. Используйте кнопки “**Выделить**” и “**Отм. выделение**” для установки списка желаемых кодов, в соответствии с вашим оборудованием. Ниже даны стандартные коды назначения ISaGRAF:

**SIMULATE:**.....Этот код предназначен для Симулятора ISaGRAF. Симулятор не может быть запущен, если эта цель не выбрана для генерации кода.

**ISA86M:**.....Этот TIC код (системно-независимый код) предназначен для ISaGRAF ядер, установленных на Intel процессорах. Тип процессора необходим только для установки последовательности байт в генерируемом коде.

**ISA68M:**.....Этот TIC код (системно-независимый код) предназначен для ISaGRAF ядер, установленных на Motorola процессорах. Тип процессора необходим только для установки последовательности байт в генерируемом коде.

**SCC:**.....Этот выбор заставляет компилятор ISaGRAF генерить исходные коды структурного языка “C”, которые будут компилироваться и линковаться с библиотеками ядра ISaGRAF для получения встроенных выполняемых кодов.

**CC86M:**.....Этот выбор заставляет компилятор ISaGRAF генерить исходные коды неструктурного языка “C”, которые будут компилироваться и линковаться с библиотеками ядра ISaGRAF для получения встроенных выполняемых кодов. Эта секция сделана для обеспечения совместимости с версиями ISaGRAF до V3.23, когда генерация кодов структурного “C” не поддерживалась.

Обратитесь к своему руководству по оборудованию, чтобы узнать тип ISaGRAF ядра, установленного на вашей PLC. Другие типы (машинные коды, исходные коды C ...) могут быть поддержаны в будущих выпусках системы разработки ISaGRAF (Workbench).



## ▬ **Обработка SFC**

Отметьте квадрат **“Использовать встроенные средства SFC”**, чтобы разрешить работу ISaGRAF SFC машины. **Этот режим следует предпочесть, так как он ведет к большей производительности.** Однако, целевая машина может быть опущена в некоторых частных реализациях целевого приложения ISaGRAF, наиболее общих заказных целевых приложениях, основанных на кодовой постобработке ISaGRAF. В этом случае вы можете быть должны удалить эту опцию и дать компилятору ISaGRAF транслировать SFC схему на нижний уровень инструкций. Обратитесь к своей документации по оборудованию для большей информации на счет этой опции.

## ▬ **Опции оптимизатора**

Ниже даны параметры, используемые Генератором Кодов ISaGRAF для оптимизации получаемых кодов, которые могут быть установлены из окна диалога **“Опции компилятора”**. Кнопка **“Умолчания”** используется для удаления всех опций оптимизации для сокращения времени компиляции.

◆ Когда установлена опция **“Оптимизация в два прохода”**, Оптимизатор Кодов запускается дважды. Оптимизация, сделанная во время второго прохода в основном менее значима, чем сделанная во время первого.

◆ Когда установлена опция **“Вычислять константные выражения”**, выражения констант вычисляются компилятором. Например, цифровое выражение **“2 + 3”** заменяется на **“5”** в получаемых кодах. Если эта опция не установлена, то выражения констант вычисляются во время работы.

◆ Когда установлена опция **“Удалять неиспользуемые метки”**, Оптимизатор упрощает систему переходов и меток программы, удаляя неиспользуемые метки или нулевые переходы.

◆ Когда установлена опция **“Оптимизировать копирование переменных”**, оптимизируется использование временных переменных (используемых для хранения промежуточных результатов). Эта опция в основном используется вместе с опцией **“Оптимизировать выражения”**. Когда эта опция установлена, Оптимизатор многократно использует результат выражений и подвыражений, которые используются в программе больше, чем один раз.

◆ Когда установлена опция **“Удалить неиспользуемый код”**, Оптимизатор подавляет коды, которые не имеют значения. Например, если запрограммировано следующее выражение: **“var := 1; var := X;”**, то соответствующий генерируемый код будет только: **“var := X;”**.

◆ Когда установлена опция **“Оптимизировать арифметические операции”**, Оптимизатор упрощает арифметические операции согласно специальным операндам. Например, выражение **“A + 0”** заменяется на **“A”**. Когда установлена опция **“Оптимизировать булевские операции”**, Оптимизатор упрощает

логические операции согласно специальным операндам. Например, логическое выражение “**A & A**” заменяется на “**A**”.



Когда установлена опция “**Строй бинарные диаграммы принятия решения**”, Оптимизатор заменяет логические уравнения (смесь операторов **AND**, **OR**, **XOR** и **NOT**) сокращенным списком операций перехода. Преобразование происходит только в том случае, если ожидаемое время последовательности переходов меньше, чем время ожидаемое для оригинального выражения.

Следующая таблица суммирует ожидаемую оптимизацию и требуемое время компиляции в соответствии с каждым параметром:

	Выигрыш(произв.)	время
Оптимизация в два прохода	xxxx	.....(*)
Оптимизировать константные выражения	xxxxxxxx	.....xxxx
Удалять неиспользуемые метки	xxxx	.....xxxxxxxx
Оптимизировать копирование переменных	xxxx	.....xxxxxxxx
Оптимизировать выражения	xxxx	.....xxxxxxxx
Удалять неиспользуемый код	xxxx	.....xxxxxxxx
Оптимизировать арифметические операции	xxxxxxxx	.....xxxx
Оптимизировать булевские операции	xxxxxxxx	.....xxxx
Строить бинарные диаграммы решений	xxxxxxxxxxxx	.....xxxxxxxxxxxx

(\*) время компиляции также умножается на 2.

### A.13.3 Генерация исходных C кодов

Система разработки ISaGRAF позволяет генерацию исходных кодов на языке “C”. В этом случае, все содержание приложения, включая описание SFC схем, определение базы данных и последовательности кодов генерится в формате языка “C”. Есть два стиля генерируемых кодов:

**CC86M** .....(C исходный код - V3.04) генерирует исходный код неструктурного “C”. Этот стиль нужно выбирать, если ваша целевая задача основана на версии ISaGRAF до 3.23.

**SCC** .....(структурный C исходный код) генерирует исходный код структурного “C”. Этот стиль нужно выбирать, если ваша целевая задача основана на версии ISaGRAF 3.23 или поздней.

Следующие два файла создаются в каталоге проектов:

**APPLI.C** ..... общие исходные коды приложения

**APPLI.H** ..... определения языка “C”

В случае генерации исходного кода структурного “C”, “.C” файлы исходников и “.H” файлы определений создаются для каждой программы приложения, в дополнение к общим файлам “**APPLI.C**” и “**APPLI.H**”. Эти файла должны быть скомпилированы и слинкованы с библиотекой ISaGRAF для получения окончательного исполняемого кода. Обратитесь к “ISaGRAF I/O development toolkit User’s Guide” для дальнейшей информации насчет рекомендуемой техники реализации.

Замечание: Некоторые отладочные возможности, такие как загрузка приложения, модификация по ходу работы и точки прерывания не доступны, когда приложение ISaGRAF скомпилировано на языке “С”.

### **А.13.4 Просмотр информации**

Меню “**Редактор**” содержит команды для просмотра различных текстовых файлов, построенных во время генерации кодов или операций проверки синтаксиса, в окне генератора кодов. Окно генератора кодов - это текстовое поле, которое содержит сообщения во время генерации кодов или операций проверки синтаксиса. Вся информация сохраняется на диске, так что она может быть изучена при помощи команд меню “**Редактор**”.

#### **⇒ Команды редактирования**

Команда “**Удалить**” используется для очистки текстового окна. Окно автоматически очищается перед каждой генерацией кодов или операцией проверки синтаксиса. Команда “**Копировать**” используется для копирования отображенного текста в Windows clipboard, так что это может быть использовано другими приложениями, такими как редакторы текста ISaGRAF.

#### **⇒ Просмотр выходных сообщений компилятора**

Команда “**Сообщения**” показывает все сообщения, отображенные в течении последних операций “**Создать**” или “**Проверить**” в текстовом окне. Это применимо ко всем сообщениям об ошибках.

Другие выборы меню “**Редактор**” позволяет пользователю контролировать вспомогательные текстовые файлы, созданные в течении проверки синтаксиса и генерации кодов. Эти файлы обычно не используются в общем проекте ISaGRAF.

### **А.13.5 Определение ресурсов**

Команда “**Ресурсы**” меню “**Опции**” позволяет пользователю определять ресурсы. Ресурсы - это любые, определенные пользователем данные (конфигурация сети, установки оборудования ...) любого формата (файл, список значений), которые должны быть соединены с генерируемым кодом, чтобы быть загруженными вместе на целевую PLC. Такие данные напрямую не управляются ядром ISaGRAF, и в основном предназначены для другого программного обеспечения, установленного на целевом PLC. Обратитесь к своему руководству по оборудованию для дальнейшей информации о доступных ресурсах.

#### **⇒ Файл определения ресурсов**

Ресурсы, определенные в “**Файле определения ресурса**” сохраняются вместе с другими файлами проекта ISaGRAF. Это чисто текстовый файл в ASCII кодах, созданный Компилятором Ресурсов ISaGRAF. Этот компилятор автоматически запускается во время построения кодов приложения. Эта секция объясняет синтаксис этого файла. Файл определения ресурсов использует лексические правила языка ST. Комментарии, начинающиеся с (\* и заканчивающиеся

символами \*), могут быть вставлены в любом месте текста. Строки разграничиваются одиночными апострофами. Обратитесь ко второй части этого руководства для большего объяснения лексических форматов, используемых для ввода числовых значений.

## ▬ Справочник по языку

Ниже дан список ключевых слов и операторов, используемых в файле определения ресурсов.

### ULONGDATA

**Значение:** Определяет ресурс, который является списком целых значений. Значения сохраняются в получаемом коде как 32 битные целые. Значения сохраняются в порядке, указанном в файле определения ресурсов. Значения должны быть разделены запятыми. Имя ресурса не должно превышать 15 символов.

**Синтаксис:** **ULONGDATA** '<имя\_ресурса>'  
**BEGIN**  
...целевой\_выбор...  
...список переменных...  
**END**

**Пример:** ULongData 'MYDATA'  
Begin  
...  
0, -1, 100\_000, (\* десятичные \*)  
16#A0B1, 2#1011\_0101 (\*шестнадцатеричн, двоичные\*)  
End

### VARLIST

**Значение:** Определяет ресурс, который является списком адресов переменных. Переменные идентифицируются своими именами в файле определения ресурсов. Адреса переменных сохраняются в получаемом коде как беззнаковые 16 битные целые. Адреса сохраняются в порядке, указанном в файле определения ресурсов. Переменные должны быть разделены запятыми. Имя ресурса не должно превышать 15 символов.

**Синтаксис:** **VARLIST** '<имя\_ресурса>'  
**BEGIN**  
...выбор\_цели...  
...список переменных...  
**END**

**Пример:** VarList 'LIST'  
Begin  
...  
Var100, MyParameter, Command, Alarm  
End

## BINARYFILE

**Значение:** Определяет ресурс Бинарный Файл. Исходные данные сохраняются в MS-DOS файле. Определение целевого ресурса завершается определением целевого имени. Конец строки символов не преобразуется Компилятором Ресурсов ISaGRAF. Имя ресурса не должно превышать **15** символов.

**Синтаксис:** **BINARYFILE** '<имя\_ресурса>  
**BEGIN**  
...выбор\_цели...  
FROM '<путь\_источника>'  
TO '<путь\_назначения>'  
**END**

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

## TEXTFILE

**Значение:** Определяет ресурс Текстовый Файл. Исходные данные сохраняются в MS-DOS файле. Определение целевого ресурса завершается определением целевого имени. Конец строки символов преобразуется Компилятором Ресурсов ISaGRAF, в соответствии с соглашениями целевой головной системы. Имя ресурса не должно превышать **15** символов.

**Синтаксис:** **TEXTFILE** '<имя\_ресурса>  
**BEGIN**  
...выбор\_цели...  
FROM '<путь\_источника>'  
TO '<путь\_назначения>'  
**END**

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

## TARGET

**Значение:** Определяет имя получаемых кодов, которые должны включать ресурс. Обратитесь к предшествующей секции (опции компилятора) для дальнейшей информации об управляемых целях. Оператор **“Target”** может появляться более, чем один раз в том же самом

блоке ресурса для выбора нескольких целей. Этот оператор не может быть использован, если указан оператор “**AnyTarget**”.

*Синтаксис:* **TARGET** '<имя\_цели>'

*Пример:*

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

## ANYTARGET

*Значение:* Определяет то, что ресурс должен быть включен во все коды, построенные Генератором Кодов. Генератор Кодов ISaGRAF может производить несколько кодов назначения во время выполнения одной команды “**Создать**”. Этот оператор не может быть использован, если указан один или несколько операторов “**Target**”.

*Синтаксис:* **ANYTARGET**

*Пример:*

```
ULongData 'MYDATA'  
Begin  
    AnyTarget  
    ...  
End
```

## FROM

*Значение:* Определяет путь источника (на PC, где установлен ISaGRAF Workbench) ресурсов **BinaryFile** и **TextFile**. Символы, используемые для разделения компонентов пути (диск, каталог, префикс, суффикс) должны соответствовать соглашениям MS-DOS.

*Синтаксис:* **FROM** '<путь цели>'

*Пример:*

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

## TO

*Значение:* Определяет путь назначения (на системе назначения) ресурсов **BinaryFile** и **TextFile**. Символы, используемые для разделения компонентов пути (диск, каталог, префикс, суффикс) должны соответствовать соглашениям системы назначения.

*Синтаксис:* **TO** '<путь цели>'

```

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

```

## Пример

Ниже дан полный пример файла определения ресурсов:

```

(* файл определения ресурсов *)

ULongData 'DATA1'          (* список значений *)
Begin
  Target 'ISA86M'          (* только для этой цели *)
  1, 0, 16#1A2B3C4D, +1, -1 (* цифровые значения *)
End

VarList 'VLIST1'          (* список переменных *)
Begin
  Target 'ISA86M'          (* только для этой цели *)
  Valve1, StateX, Command, Alrml (* имена переменных *)
End

BinaryFile 'FILE1'        (* ресурс бинарный файл *)
Begin
  AnyTarget                (* для любой цели *)
  From 'c:\user\updatef.bin' (* исходник на PC *)
  To 'updatef.cfg'         (* файл назначения на PLC *)
End

TextFile 'FILE2'          (* ресурс текстовый файл *)
Begin
  Target 'ISA68M'          (* только для этой цели *)
  From 'c:\nw\nwbd.txt'    (* исходный файл на PC *)
  To '/nw/dat/nwbd'        (* файл назначения на PLC *)
End

```

## Компиляция ресурсов

Если ресурсы были введены в файл определения ресурсов, окно диалога появится в конце генерации кодов ISaGRAF. Нажмите кнопку **“Запустить компилятор”** для запуска компилятора ресурсов. Выходные сообщения и ошибки будут отображены в основном окне управления. Нажмите **“Выход”** для избежания компиляции ресурсов. В этом случае ресурсы не будут добавлены в коды ISaGRAF.

## Реализация

Число ресурсов, размер строк данных и файлов ISaGRAF не ограничивает. Ресурсы сохраняются в конце генерируемого кода с каталогом ресурсов. Ниже дан формат (используя C нотацию) формата каталога ресурсов:

```

RESOURCE:
{
  long nbres;           /*количество определенных ресурсов*/
  {
    char name[16];      /* имя ресурса */
    long type;          /* тип данных ресурса */
    long size;          /* точный размер блока данных */
    void *data;
    char *path_offset; /* указывает на строку */
  } /*nb записей */
}

```

Ниже даны возможные значения поля "type":

- 1 = бинарный файл
- 2 = текстовый файл
- 3 = ulong данные (в этом случае поле path\_offset не используется)
- 4 = список переменных (в этом случае поле path\_offset не используется)

Для текстовых файлов конец строки символов преобразуется компилятором ресурсов в зависимости с соглашениями системы назначения. Все указатели - это 32 битные смещения от адреса соответствующей структуры. Все имена ресурсов и пути - это строки, заканчивающиеся на NULL. Пути и данные следуют за каталогом ресурсов.



## A.14 Перекрестные ссылки

Система разработки ISaGRAF (workbench) включает редактор перекрестных ссылок, который обеспечивает пользователя полным просмотром объявленных переменных в программах проекта, и мест, где они используются. Цель перекрестной ссылки - это перечисление всех переменных, объявленных в проекте, и локализация, в исходных кодах каждой программы, частей исходного кода, где эти переменные используются. Перекрестные ссылки очень полезны для глобального просмотра жизненного цикла одной переменной. Они помогают локализовать побочные эффекты и сократить время понимания проекта во время его построения. Перекрестные ссылки могут также быть использованы для глобального просмотра всего словаря проекта, так что легко находятся неиспользуемые переменные и оценивается сложность проекта.

Список слева показывает объявленные объекты проекта (программы, переменные и определенные слова), и элементы библиотеки (функции и блоки функций), на которые есть ссылки в проекте. Список справа показывает вхождения в программах объекта, выбранного в данный момент в первом списке.

Описание вхождений включает имя программы, количество шагов FC или SFC, переход или тест, плюс номер строки для текстовых языков или координаты для LD или FBD диаграмм. Для диаграмм quick LD, описание завершается номером штанги. Если переменная используется, как выход, (виток) за номером штанги следует символ звезды ("\*").

Установить опцию "**Показать неиспользованные переменные**" из меню "**Опции**", чтобы показать, также в основном списке переменные, которые не используются в прикладных программах.

### **Выбор типа объекта**

Так как проект может включать большое количество объявленных объектов, то используется combo box в меню средств редактора для выбора типа объектов, которые должны быть отображены в окне. Это позволяет пользователю иметь доступ к выборочной информации.

Каждый раз, когда перекрестные ссылки заново вычисляются, выбор сбрасывается в положение "**Все**" для того, чтобы представить весь список.

### **Вычисление перекрестных ссылок заново**

Команда "**Файл / Пересчитать**" может быть использована в любое время для обновления перекрестных ссылок, в соответствии с изменениями, введенными в других окнах редактирования ISaGRAF.

### **Экспорт перекрестных ссылок**

Команда "**Файл / Экспортировать**" используется для записи полного списка перекрестных ссылок в текстовый файл ASCII. Этот файл затем может быть открыт другими приложениями, такими как Windows NotePad или текстовыми редакторами.



### **Словарные ошибки**

Команда **“Редактор / Ошибки словаря”** отображает в окне диалога список ошибок обнаруженных во время загрузки словаря проекта.



### **Статистика**

Команда **“Инструменты / Статистика”** отображает в окне диалога число объектов и переменных, объявленных в проекте, следуя типу и атрибутам переменных. Частное приложение этой команды - это определение количества переменных В/В объявленных в проекте, для того, чтобы быть уверенным, что он может быть скомпилирован, если установлена ограниченная версия ISaGRAF Workbench.



### **Поиск в списке объектов**

Команда **“Редактор / Поиск”** позволяет пользователю напрямую выбирать объект в списке редактора. Искомый объект не может быть найден, если он не был указан в списке (когда используется выборочное отображение информации). Перед поиском объекта рекомендуется активизировать выбор **“Все”** в меню средств.



### **Открытие программы**

Список справа содержит вхождения выбранного объекта в исходных файлах и соединении В/В открытого проекта. Команда **“Редактор / Открыть программу”** позволяет пользователю непосредственно открывать программу там, где этот объект появился. Также, можно ) открыть соответствующую программу двойным щелчком мышки на ее вхождении (в списке вхождений).

## A.15 Использование графического отладчика

ISaGRAF включает полный графический и символьный редактор. Команда “Отладка” окна управления программ запускает отладчик для управления приложением, загруженным на целевую PLC. В этом режиме отладчик общается с целевой системой через аппаратную связь. Команда “Симуляция” окна управления программ одновременно запускает отладчик и полный симулятор целевой системы. Это позволяет пользователю тестировать его приложение, когда целевая система В/В еще не завершена. Окно отладчика содержит команды для управления всем приложением.

Если, при запуске отладчика, приложение на целевой PLC то же, что и на workbench, то автоматически открывается **окно управления программами** в режиме отладки. Команды этого окна могут быть использованы для открытия других окон ISaGRAF (графического и текстового редакторов, словаря, списков переменных, соединений В/В ...). Все окна, открытые во время сессии отладки функционируют в **“режиме отладки”**, означающим то, что команда редактирования запрещена. Отображенные компоненты программ (шаги, переходы, переменные ...) показаны с их текущим состоянием или значением. Двойной щелчок мышкой по объекту изменяет его статус или значение в целевом приложении.

Когда отладчик запускается в режиме симуляции, связь с целевой системой ISaGRAF останавливается. Отладчик взаимодействует только с окном симуляции. Так как целевая система не существует в данном режиме, то команды **“загрузить”**, **“запустить приложение”** или **“остановить приложение”** не доступны в меню отладчика.

### A.15.1 Окно отладчика

Окно отладчика содержит информацию только о завершеном состоянии приложения. Оно связано с другими окнами ISaGRAF, создавая полную интерактивную систему отладки. Обнаруженные ошибки выполнения отображаются в нижней области окна отладчика. Команды из меню **“Опции”** используются для скрытия, показа или удаления списка ошибок.

Панель управления (область под меню отладчика) показывает глобальное состояние целевого приложения, а также информацию о времени выполнения цикла. Список возможных состояний:

**Регистрация:** ..... Отладчик устанавливает соединение с целевой системой.

**Нет связи:** ..... Отладчик не может соединиться с целевой системой. Убедитесь, что соединительный кабель и параметры коммуникации правильны.

**Нет приложения:** ..... Соединение в норме, но приложение ISaGRAF в данный момент отсутствует на целевой системе. Загрузите приложение.

**Приложение активно:**... Соединение в норме и действующее приложение существует на целевой системе. Отладчик в данный момент устанавливает связь с этим приложением, если оно то же самое, что и на Workbench...

**Работа:** ..... Целевое приложение в режиме “Реальное время”.  
**Стоп:** ..... Целевое приложение в режиме “Пошаговый”.  
**Точка Останова:** ..... Целевое приложение в режиме “Пошаговый”, потому что достигли точки останова.  
**Ошибка:** ..... Сбой в целевом приложении из-за серьезной ошибки.

Информация о времени выполнения цикла показана ниже:

**Разрешено:** ..... запрограммированное время.  
**Текущий:** ..... точное время последнего завершенного цикла.  
**Максимальный:** ..... максимальное время, выявленное с начала запуска приложения.  
**Переполнение:** ..... число циклов выполнения, выявленных с временем большим, чем запрограммированное время.

Все значения даны в миллисекундах. Значения времени не отображаются, если отладчик запущен в режиме симуляции.

## A.15.2 Управление приложением

Меню “**Файл**” и “**Управление**” содержат команды для установки и управления текущего редактируемого приложения ISaGRAF на целевой системе ISaGRAF.

Замечание: Некоторые из этих команд недоступны во время симуляции, потому что приложение, обрабатываемое симулятором, автоматически устанавливается ISaGRAF Workbench.



### **Остановка целевого приложения**

Команда “**Файл / Остановить приложение**” останавливает выполнение приложения активного в данный момент на целевой системе ISaGRAF.

### **Активизация целевого приложения**

Команда “**Файл / Запустить приложение**” запускает на выполнение приложение, существующее на целевой системе. Когда приложение загружено, то оно автоматически запускается, так что нет необходимости использовать команду “**Запустить приложение**”. Команда “**Запустить приложение**” обычно используется после команды “**Остановить приложение**”.

Замечание: Целевое приложение должно быть остановлено (дезактивировано) перед тем, как станет возможным загрузить новое приложение.



### **Загрузка приложения**

Команда “**Файл / Загрузить**” используется для загрузки кода приложения на целевую систему. Выберите тип кода, который будет загружаться, в соответствии с целевым системным процессором и опциями приложения.

### **Отображение номера версии**

Команда “**Файл / Номер версии**” используется для отображения идентификации и Workbench, и целевого приложения. Приложение Workbench - одно приложение,

открытое на ISaGRAF Workbench. Целевое приложение - одно приложение, выполняемое на целевой ISaGRAF PLC. Отображаются следующие позиции:

**Версия:** ..... Это номер версии кода приложения. Этот номер был вычислен генератором кодов.

**Дата:** ..... Эта позиция показывает дату и время создания кодов.

**CRC:** ..... Это контрольная сумма вычисленная вместе с содержимым таблицы символов. Это число было подсчитано генератором кодов. Его значение зависит от содержимого словаря переменных.

Замечание: Команда “**Номер версии**” также доступна во время симуляции. В режиме реальной отладки эта команда не может быть использована, если целевое PLC не подключено.



### **Модификации во время работы**

Команда “**Файл / Изменить приложение**” позволяет пользователю осуществить “модификацию на ходу” запущенного целевого приложения. Эта команда детализирована в следующих разделах этой главы. Она недоступна, когда отладчик используется в режиме симуляции.



### **Режим Реального Времени (Реальное время)**

Команда “**Управление / Реальное время**” недоступна, когда нет активного приложения. Она устанавливает целевое приложение в нормальный режим “реального времени”: Нормальный режим: циклы выполнения запускаются с запрограммированным временным циклом .



### **Режим Цикл за Циклом (Пошаговый режим)**

Команда “**Управление / Пошаговый режим**” недоступна, когда нет активного приложения. Она устанавливает целевое приложение в нормальный режим “цикл за циклом”: В этом режиме циклы выполняются один за одним, в соответствии с командами “**Выполнить один цикл**”, заданными пользователем из меню отладчика.



### **Выполнить один цикл**

Когда целевое приложение находится в режиме цикл за циклом, команда “**Управление / Выполнить один цикл**” запускает выполнение одного цикла.



### **Временной цикл**

Команда “**Управление / Изменить временной цикл**” позволяет пользователю изменять запрограммированный временной цикл. Это время обозначено как “**Разрешено**” в окне управления отладчиком. Режим “**Пошаговый режим**” должен быть установлен перед изменением временного цикла. Временной цикл вводится как целое число в миллисекундах.



### **Удаление всех точек останова**

Команда **“Управление / Очистить все токи останова”** удаляет все текущие точки останова (встреченные или еще активные) во всем приложении. Существующие точки останова автоматически не удаляются, когда закрывается окно отладчика.

### ▬ **Разблокирование переменных В/В**

Команда **“Управление / Открыть все переменные ВВ”** разблокирует все переменные В/В, в настоящий момент заблокированные в приложении. Когда переменная В/В заблокирована, состояние входа или выхода не изменяется в соответствующем устройстве В/В. Переменные, подключенные к В/В, могут быть еще записанными приложением или отладчиком. Текущие заблокированные переменные В/В автоматически не разблокируются, когда закрывается окно отладчика.

## **A.15.3 Опции**

Меню **“Опции”** содержит опции для управления информацией, выводимой в окне отладчика:

### ▬ **Параметры связи**

Временные параметры связи могут регулироваться когда отладчик активен. Здесь может быть установлен только таймаут связи. Другие параметры связи (скорость передачи, паритет ...) должны быть установлены из меню **“Отладка”** окна Управления Программ.

**“Таймаут связи”** - это время, оставленное для целевой системы для начала ответа по запросу workbench. **“Время обновления”** - это период времени, требуемый для запроса на чтение, который будет посылаться отладчиком для обновления данных в открытом окне.

Все временные значения отображаются и вводятся как целые числа в миллисекундах. Временные параметры связи не могут быть установлены, когда отладчик используется в режиме симуляции.

### ▬ **Опции экрана**

Опция **“Показать временной цикл”** позволяет пользователю скрыть или показать значения временного цикла. Если эта опция установлена, то отображаются и обновляются все компоненты временного цикла (разрешено, текущий, максимум, переполнения). Запрещение этой опции сокращает затраты на связь отладчика.

Когда установлена опция **“Показать ошибки”**, обнаруженные ошибки выполнения перечисляются в нижней части окна отладчика. Если это опция запрещена, то список ошибок закрыт. Удаление этой опции сокращает затраты отладчика на отображение и связь. Команда **“Опции / Очистить ошибки”** очищает список ошибок выполнения, в данный момент отображенных в окне отладчика.

Команда **“Опции / Минимизировать окно”** уменьшает размер окна отладчика, так что оно отображается как маленькая, всегда сверху, панель, содержащая только состояние приложения и графические кнопки для наиболее общих используемых команд.

## A.15.4 Команды записи

Символьный отладчик ISaGRAF предлагает много команд для изменения значения или состояния компонент приложения. Выбор компоненты для изменения производится двойным щелчком мышкой на ее имени, или на ее рисунке в окне редактирования, когда открыто окно отладчика.

### ▣ **Переменные**

Состояние переменной изменяется двойным щелчком мышкой по ее имени в одном из следующих окон:

- Словарь
- Списки переменных или временные диаграммы
- LD или FBD Программы
- Соединения В/В

Следующие команды предлагаются в окне диалога отладки:

- Запись переменной в новое значение
- Блокирование переменной (только для переменных В/В)
- Разблокирование переменной (только для заблокированных переменных В/В)
- Запуск или остановка переменной таймера (установить режим автоматического обновления)

Символьные значения используемые для представления логических значений **TRUE** и **FALSE** - это строки, определенные для этих специальных логических переменных в словаре. Аналоговое значение, определенное для команды **“Писать”**, должно быть введено в целом или вещественном формате, в соответствии с определением переменных в словаре. Строка, определенная для команды **“Писать”** для сообщения, не может быть больше, чем размер сообщения, подключенного к этой определенной переменной, в словаре.

### ▣ **Объекты SFC**

Для просмотра операции управления в **программе SFC** во время отладки приложения, используются команды меню **“Файл”** в окне Управления Программ. Программа SFC должна быть выбрана из списка программ. Доступны следующие команды:

**Стартовать программу SFC:** Разрешает выбранную программу путем помещения SFC маркера в каждый ее начальный шаг.

**Убить программу SFC:** Убивает выбранную программу путем удаления всех ее существующих маркеров.

**Заморозить программу SFC:** Удаляет все существующие маркеры выбранной программы и сохраняет их расположение.

**Перезапустить программу SFC:** Перезапускает замороженную (“замороженная”) программу путем восстановления маркеров, которые были удалены командой **“Заморозить”**.

Для дочерних программ эти команды соответствуют функциям **“GSTART”**, **“GKILL”**, **“GFREEZE”** и **“GRST”** на языке программирования.

Операция управления может быть увидена в **шаге SFC**, когда отлаживается приложение, путем двойного щелчка мышкой на ее графическом представлении в

окне редактирования SFC. Следующие команды доступны в окне диалога отладки:

- Установка точки останова в шаге **активизации**
- Установка точки останова в шаге **деактивизации**
- **Удаление** точки останова добавленной в шаг

Замечание: Активизация и деактивизация точек останова не может быть добавлена в тот же самый шаг.

Операция управления может быть увидена в **переходе SFC**, когда отлаживается приложение, путем двойного щелчка мышкой на ее графическом представлении в окне редактирования SFC. Следующие команды доступны в окне диалога отладки:

- Добавить точку останова в месте перехода
- Удаление точки останова добавленной к переходу
- Ручная очистка перехода (передвинуть или добавить маркер)

**Условное удаление:** маркер создается на шаге следующим за переходом. Маркеры, существующие в предыдущих шагах удаляются. **Безусловное**

**удаление:** маркер создается на шаге следующим за переходом. Маркеры, существующие в предыдущих шагах не удаляются.

## A.15.5 Изменение по ходу работы

Свойство “Изменение по ходу работы” (“On line modification”) позволяет пользователю изменять приложение во время работы процесса. Это иногда необходимо для химических процессов, где любое прерывание может подвергнуть опасности продукцию или безопасность. Этой функцией следует пользоваться очень осторожно. ISaGRAF может быть не в состоянии распознать все возможные конфликты, созданные операциями, определенными пользователем, при этих изменениях по ходу работы.

### ⇒ **Последовательности кодов**

Так как ISaGRAF предоставляет много возможностей для доступа к переменным, программам или платам В/В из отладчика, функция “Изменение по ходу работы”, описанная здесь, применяется только для изменения последовательности кодов. Последовательность кодов - это полный набор ST, IL, LD или FBD инструкций, выполненных в строке. В программе “начало цикла” (“beginning of cycle”) или “конец цикла” (“end of cycle”), последовательность кодов - это полный список инструкций, написанных в программе. В SFC программе последовательность кодов - это Уровень 2 программирования одного шага или перехода. “Изменение по ходу работы” состоит из замены одной или более последовательности кодов, без остановки цикла выполнения PLC. Так как управление маркерами SFC очень критично, то **невозможно изменение структуры SFC, добавление, перенумерация или удаление шага, перехода или SFC программы.**

### ⇒ **Переменные**

Так как база данных переменных - это очень критическая часть приложения, она может быть доступна в любое время другими процессами (на многозадачной PLC). Также возможно изменение значения переменных из отладчика. Следовательно, **ISaGRAF не позволяет пользователю добавлять,**



**переименовать или удалять переменные** по ходу работы приложения. В любом случае возможно изменить способ использования переменной в приложении. Также возможно резервировать “неиспользуемые” внутренние или V/B переменные в первой версии приложения, так что будущие изменения могут использовать их.

В базе данных ISaGRAF существуют различные стили переменных. Ограничения распространяются на всех:

#### *- Объявленные переменные*

Переменные однажды объявленные в словаре ISaGRAF. Они не могут быть изменены или переименованы по ходу работы. Рекомендуется объявлять и инициализировать в приложении некоторые дополнительные переменные даже если они не используются сейчас. Такие дополнительные переменные позволят в дальнейшем модифицировать приложения без изменения контрольной суммы.

#### *- Экземпляры функциональных блоков*

Каждый экземпляр функционального блока на "С" или IEC соответствует данным сохраненным в базе данных реального времени ISaGRAF. Когда экземпляр функционального блока добавляется или удаляется, Модификация по Ходу Работы становится невозможной. Так, что лучше работать в ST с экземплярами FB объявленными в словаре, чем добавлять блоки (которые будут соответствовать новым автоматически объявленным экземплярам) в Quick LD или FBD диаграммах. Также, любая модификация в определении функционального блока в библиотеке ISaGRAF приведет к невозможности изменений по ходу работы.

#### *- Шаги*

Каждый шаг SFC соответствует куску данных, где хранятся динамические атрибуты шага SFC (его время активности и флаг). Добавление или удаление шагов SFC изменяет базу данных и запрещает изменение по ходу работы.

#### *- Скрытые переменные размещенные компилятором*

Компилятор ISaGRAF генерирует “скрытые”, временные переменные чтобы разрешить сложные выражения. В некоторых случаях, изменение приложения может привести к отличному набору невидимых временных переменных, и это ведет к невозможности изменения по ходу работы. Чтобы избежать этой ситуации, вы можете добавить следующие вхождения в файл ISA.INI, для того чтобы заставить размещать минимальное количество временных переменных для каждой программы, даже если они не используются для компиляции первой версии приложения. Значения даны для примера:

```
[DEBUG]
MNTVboo=8      ; для булевских
MNTVana=4      ; для integer и real
MNTVtmr=4      ; для timer
MNTVmsg=2      ; для message
```

Когда такие установки написаны в ISA.INI, компилятор выводит предупреждение если новая компиляция приложения ведет к к большому количеству размещенных временных переменных.

## ⇒ **Входы и выходы**

Так как ISaGRAF система очень открытая, требуемые изменения должны быть воплощены OEM, используя определенные свойства соответствующего оборудования. Система ISaGRAF **не позволяет пользователю добавлять, подсоединять или удалять переменные В/В, или изменять описание платы В/В** по ходу работы. Такие операции, как изменение параметров платы и запираение каналов В/В, возможны используя стандартные свойства OEM и функцию “OPERATE”.

## ⇒ **Операции во время работы**

Изменение запущенного приложения состоит из следующих операций:

- изменение исходного кода приложения на workbench
- генерация нового кода приложения
- загрузка нового кода приложения, используя команду “изменить” вместо “загрузить”
- переключение со старого приложения на новое между циклами выполнения PLC, используя команду “Выполнить изменение”.

Эта процедура гарантирует, что целевой PLC всегда имеет полное и надежно работающее приложение, и позволяет пользователю управлять синхронизацией эталонных операций очень безопасным и эффективным путем. Это также позволяет пользователю изменять проект при любой возможности. Несмотря на сам процесс, “изменение по ходу работы”, в основном, то же самое, что и нормальный набор команд “остановить, запустить и загрузить”. Отличия состоят в том, что состояния переменных не теряются и время переключения очень мало (обычно в течении 1 или 2 циклов). Во время переключения переменные не изменяются, и **все внутренние, входные и выходные переменные сохраняют то же самое значение**, и до, и после изменения приложения. Во время переключения никакие действия не выполняются и **SFC маркеры не передвигаются**.

## ⇒ **Требования по памяти**

Для поддержки способности “изменения по ходу работы”, целевое PLC должно иметь свободное место в памяти для возможности сохранения измененной версии кода приложения. Обе версии кода приложения должны храниться во время операции переключения.

## ⇒ **Ограничения**

Как было описано ранее, разрешены только изменения последовательности кодов. Определение переменных, параметров приложения и соединений В/В не может быть изменено. Когда загружается измененная версия приложения, ISaGRAF делает сравнение между измененным приложением и запущенным для определения любых небезопасных изменений. Если переключение кажется опасным или невозможным, то генерируется ошибка загрузки. Одна из защит

выполняемых ISaGRAF - это сравнение символьной таблицы контрольной суммы, так что определяется любое изменение имен переменных, программ или элементов SFC. Если шаг активен, когда совершается переключение, то не сохраненные действия (N) будут потеряны. Действия активизации нового шага не выполняются. Действия, выполненные при деактивизации шага, - это действия, переходящие в код нового приложения. Если переход достоверен, когда происходит переключение, то его приемная часть обновляется. Новый загруженный код приложения не дублируется на PLC. Дубль - это версия, которая была предварительно загружена стандартными командами загрузки.



## Операции

Для обновления кода запущенного приложения, следующие операции должны быть выполнены:

- Перед тем, как сделать любое изменение запущенного приложения, настоятельно рекомендуется сделать копию текущего проекта под другим именем. Изменения могут быть произведены на копии.
- Перед редактированием любой программы, пользователь должен проверить, что опция средств редактирования **“изменить дневник”** установлена для упрощения будущего сопровождения программы.
- Когда одна или более последовательностей были изменены (без изменения структур SFC и иерархии программы), коды нового приложения должны генериться на workbench перед загрузкой.
- Используя отладчик внутри старого проекта, пользователь должен соединиться с целевым PLC и выполнить любую операцию, которая может сделать обновление приложения быстрее или более безопасно.
- Используя отладчик внутри нового проекта, пользователь должен соединиться с целевым PLC. Если имя приложения изменено, то целевая база данных не будет доступна. Пользователь должен запустить команду **“Файл / Изменить приложение”**.
- Измененное приложение загружается выбором опции **“обновить потом”**. Это может немного замедлить PLC во время пересылки.
- Когда загрузка завершена пользователь может запустить команду **“Файл / Выполнить изменение”** для разрешения переключения в наиболее подходящий момент. Переключение будет в течении 1 или 2 циклов.
- Когда переключение будет корректно выполнено, то будут показаны программы запущенного измененного приложения. Если нет, то значит существующее запущенное приложение осталось как было.

### A.15.6 Обмены DDE

Отладчик ISaGRAF содержит DDE (Динамический Обмен Данными) сервер. Петля уведомления (“an advise loop”) может быть установлена между отладчиком ISaGRAF и другими приложениями, для того чтобы динамически отображать текущее значение переменных в не ISaGRAF приложениях.

Только транзакции “advise” или “poke” поддерживаются DDE сервером отладчика ISaGRAF. Вы можете использовать транзакцию “request” только для переменных, уже просматриваемых в петле уведомления. Другой DDE сервис, такой как “execute” недоступен. Когда петля уведомления установлена для переменной, значение этой переменной обновляется в приложении клиента каждый раз, когда

оно изменяется. Просматриваться могут переменные любого типа. Идентификация динамической связи включает следующие имена:

Service name:..... "ISaGRAF"

Topic name:..... Имя ISaGRAF проекта

Item name:..... Имя переменной

Если переменная является локальной для программы, то за ее именем должно следовать имя ее родительской программы, написанное в скобках, со следующим синтаксисом:

**variable\_name(program\_name)**

DDE сервер отладчика ISaGRAF предназначен для приложений ISaGRAF, в текущий момент просматриваемых отладчиком. До 256 переменных могут быть просмотрены сервером ISaGRAF. DDE сервер может быть использован, когда отладчик ISaGRAF запущен либо подключенном режиме, либо в режиме симуляции. Период обновления - это период, установленный для связи между отладчиком и целевой системой ISaGRAF или симулятором.

## A.16 Списки переменных

Команда “Списки переменных” в меню “Списки шпиона” окна Отладчика позволяет пользователю строить отдельные списки переменных, которые обновляются их текущими значениями. Списки строятся во время отладки приложения. Списки могут быть сохранены на диске и открыты снова во время другой сессии отладки. Список может содержать до 32 переменных. Переменные разного типа могут быть смешаны в одном и том же списке. Глобальные и локальные переменные могут быть вставлены в список. Список переменных предназначен для одного проекта. Списки переменных очень полезны для функционального тестирования приложения. Они позволяют пользователю наблюдать изменения ограниченной части контролируемого процесса, независимого от соответствующих исходных кодов в программах приложения. Списки переменных также полезны во время отладки текстовых ST и IL программ. Пользователь может легко сгруппировать в список набор переменных, используемых в программе, для того чтобы контролировать или отображать выполнение запрограммированных инструкций. Для каждой переменной списка ISaGRAF показывает ее имя, текущее значение и текст комментария. Размер колонок может быть изменен путем перетаскивания разделительной линии в панели заголовка списка.

### Сохранение списков на жесткий диск

Команды меню “Файл” используются для создания, открытия и сохранения списков переменных. ISaGRAF не ограничивает число списков для одного проекта. Правила, показанные ниже, должны выполняться при наименовании списков переменных, которые будут сохраняться на диск:

- имя не должно превышать 8 символов
  - первый символ должен быть буквой
  - следующие символы могут быть буквами, цифрами или символом подчеркивания
  - Верхний и нижний регистры в наименовании списков не различаются
- Редактор списка не может отобразить больше, чем один список переменных за раз в одном и том же окне. Однако, редактор списка может быть запущен более, чем один раз, для просмотра различных списков одновременно.



### Вставка переменных в список

Команда “Редактор/ Вставить” вставляет другую переменную в список. Имя переменной выбирается в списке объектов, определенных в словаре проекта. В этом случае пользователь не должен вводить ручную идентификатор. Переменная вставляется перед переменной, выбранной в данный момент в списке. Список не может содержать более 32 переменных. Одна и та же переменная не может появляться более одного раза в том же самом списке.



### Изменение выбранной переменной

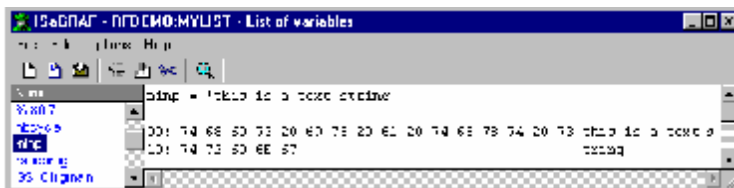
Команда “Редактор/ Модифицировать” замещает выбранную переменную другой переменной. Вы можете также использовать команду “Вырезать” для удаления выбранной переменной из списка.



## Показать Дамп

В любой момент вы можете изменить режим просмотра между списком и просмотром "Дампа". Нажмите кнопку "увеличение" на панели инструментов или используйте команду "**Опции / Дамп**" для изменения режима просмотра.

В режиме "Дамп", отображается только одна переменная. Ее значение отображается в числовом/символьном формате вверху окна, и также появляется двоичный дамп формат. Этот режим позволяет вам просматривать шестнадцатеричные значения каждого байта переменной.



Экран "Дамп" полезен для просмотра сообщений содержащих печатные символы.

## A.17 Отладка программ ST и IL

Во время симуляции или отладки программ ST и IL, никакие модификации не могут быть введены в программный текст.

**IL** Для программ IL, инструкции сформатированы в виде списка. Текущее значение переменной использующейся в инструкции появляется в той же строке. Вы можете дважды щелкнуть по инструкции, чтобы изменить значение переменной.

**ST** Для программ ST, Окно Списка встроено в окно редактора. Вы можете изменять размер просмотров перетаскивая мышью линию разделения между ними.

Для каждой переменной списка, ISaGRAF показывает ее имя, текущее значение и текст комментария. Размер колонок может быть изменен путем перетаскивания разделительных линий.

### **Сохранение списков на жесткий диск**

Команда **"Файл / Сохранить список"** сохраняет списки переменных на диске, под тем же именем что и редактируемая программа. Этот список будет автоматически подгружаться каждый раз, когда программа ST или IL открывается в режиме отладки. Этот список может быть свободно открыт и модифицирован с использованием средств Списка запущенных с помощью команды **"Инструменты / Список переменных"** окна отладки.



### **Вставка переменных в список**

Команда **"Редактор/ Вставить переменную"** вставляет другую переменную в список. Имя переменной выбирается в списке объектов, определенных в словаре проекта. В этом случае пользователь не должен вводить вручную идентификатор. Переменная вставляется перед переменной, выбранной в данный момент в списке. Список не может содержать более **32** переменных. Одна и та же переменная не может появляться более одного раза в том же самом списке.



Если имя переменной в тексте ST выделено, нажмите эту кнопку на панели инструментов или запустите команду **"Редактор/ Подсмотреть выделение "**, чтобы напрямую послать переменную во встроенный список шпиона.



### **Изменение выбранной переменной**

Команда **"Редактор/ Изменить переменную"** замещает выбранную переменную другой переменной. Вы можете также использовать команду **"Вырезать"** для удаления выбранной переменной из списка.

## A.18 Прожектор

С помощью средства ISaGRAF «прожектор» пользователь может определить списки наблюдения, которые могут показывать либо графические картинки либо списки во время отладки. Графические символы должны быть связаны с переменными ISaGRAF. Картинки могут быть определены и анимированы в процессе работы.

Для управления значением переменной щелкните дважды по соответствующему графическому символу или схеме в списке. Или нажмите ENTER когда она выбрана.

Вы можете также закрыть документ (запретить модификацию) используя команду «Файл/Закрыть». Когда документ закрыт Вы все же можете воздействовать на переменную при помощи двойного щелчка на ее символе.

### A.18.1 Построение графической схемы

Схема состоит из фоновых картинок (битмапов или метафайлов), и набора графических символов, которые будут анимированы во время отладки. Чтобы ввести схему, следующие операции должны быть выполнены: введите фоновые картинки, введите графические символы, свяжите объекты и переменный проекта.



#### *Фоновые картинки*

Фоновые картинки - это "битмапы" (.BMP) или «метафайлы» (.WMF). Количество картинок включенных в схему не ограничено. Картинки можно двигать или менять размер в графической схеме. Они не появляются в схеме списка. Картинки строятся с помощью других средств. Прожектор не включает средства рисования. Команда "Опции / Цвет фона" используется для выбора цвета пустого пространства в графической схеме.

Замечание: Битмапы требуют большого количества памяти. Настоятельно рекомендуется тщательно выбирать размер картинки, и ограничивать неиспользуемое пространство.

123

#### *Текст*

Символ «текст» это текст написанный в прямоугольнике. В прямоугольнике выводится значение присоединенной переменной. Таким образом, символ может быть связан со строкой сообщения переменной.

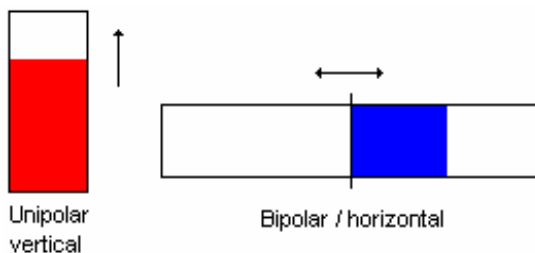
Прямоугольник, в который выводится текст может быть цветным или прозрачным. Шрифт символа подстраивается под размер.



#### *Однополярные и биполярные колонки*

Колонка - это закрашенный прямоугольник, которая представляет собой численное значение с присоединенной переменной. Колонка может быть горизонтальной или вертикальной. Однополярная колонка может расти только в одном направлении. Биполярная колонка может расти в положительном или в отрицательном направлении, в зависимости от присоединенной переменной.

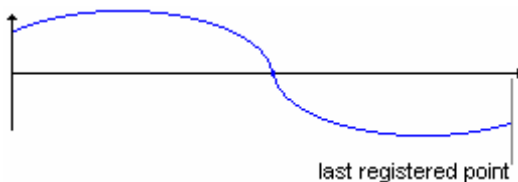




## Кривые

В документ можно встраивать кривые. Кривая показывает предысторию присоединенной переменной. Хотя это не точное средство измерения, оно может дать полезную отладочную информацию о синхронизации переменных.

Кривая запоминает 200 последних значений переменных. Количество измерений не меняется, когда изменяется размер графической схемы



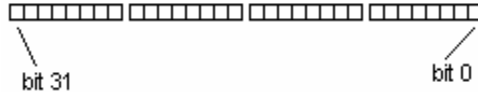
## Булевские иконки

Символ «булевская иконка» используется для отображения бинарных состояний. Один файл (.ICO) определяется для состояния FALSE или 0. Другой - для ненулевого состояния. Так как Прожектор не включает редактор иконок, файлы должны быть подготовлен с помощью других средств.



## Битовые поля

Символ «битовое поле» представляется на графической панели с помощью 32 битового целого числа. Менее значимые биты располагаются справа. Не рекомендуется использовать битовые поля для других типов данных таких как аналоговые значения.



### **Выбирать, двигать или менять размер**

Выбор графических объектов требуется для большинства команд редактирования. Прожектор позволяет один или более существующих объектов на схеме. Для выбора объектов, нужно выбрать "выбор" (кнопка со стрелкой) на полоске средств. Для выбора одного объекта нужно просто щелкнуть по символу. Для выбора нескольких объектов протащите мышку и выберите прямоугольную область. Все графические объекты, которые пересекаются с прямоугольной областью будут помечены как "выбранные". Выбранный объект нарисован с маленьким черным квадратом вокруг графического символа.

Для того, чтобы переместить объекты, их нужно сначала выбрать. Затем тащите их с помощью мышки в другое место.

Чтобы изменить размер объекта, его нужно сначала выбрать. Затем поместите стрелку на маленький прямоугольник на границе выбора и тащите его до нужного размера.



### **Группировать символы / разбивать группы**

Вы можете объединять символы в группы и работать с ними как с одним символом. Для того, чтобы создать группу, выберите символы в графической схеме и запустите команду «Редактор/ Группа». Команда «Редактор/ Разбить» используется для того, чтобы рассматривать символы выбранной группы как отдельные.

Группа может содержать картинку. Группа, также может содержать другую группу. Когда символы сгруппированы их стиль больше не меняется.

## **A.18.2 Схема списка**



В любой момент Вы можете переключаться между графической схемой и схемой списка. Вы можете также использовать команду "Опции / Список - Графическая схема".

В схеме списка символы представлены классическими коробками. Высота каждого символа рассчитывается в соответствии со стилем. Картинки (битмапы и метафайлы) не видны на схеме списка. Имеется возможность выбора на схеме списка и должно использоваться для установки стиля символа или изменения значения переменной.



Вы можете переупорядочить символы в списке используя команду "Редактор/ Двигать в списке". Символы, которые надо сдвинуть должны быть выбраны в списке.

### A.18.3 Определение стиля символа

Графический стиль и установки существующего символа могут быть изменены, если щелкнуть по его символу или запустить команду «Редактор/ Установить стиль». Диалог «Стиль» также открывается, когда новый символ добавляется в документ. Он объединяет следующую информацию:

#### ⇒ **Графический стиль и установки:**

Стиль отображения (текст, колонка, кривая...) символа может быть изменен динамически. Цвета могут быть настроены с помощью соответствующих коробок. Если стиль - "булевская иконка", должен быть определен путь соответствующий .ISO файлу. Используйте "..." кнопки рядом с этими контролями, чтобы посмотреть иконки существующие на диске.

#### ⇒ **Масштаб:**

Это максимальное значение, которое может быть колонками или кривыми. Для биполярных колонок и кривых, используется одно и то же абсолютное значение на положительной и отрицательной осях.

#### ⇒ **Имя переменной:**

Когда поле "Имя" является активным, кнопка "..." позволяет пользователю найти имя уже объявленной в словаре.

#### ⇒ **Заголовок:**

Заголовок может быть представлен рядом с графическим символом или в графической схеме. Вы можете настроить расположение заголовка (верх, низ, лево или право) и его содержание. Заголовок может быть комбинацией имени переменной и его значения, сформатированного как текст. Настройка заголовка не влияет на схему списка.

#### ⇒ **Командная переменная:**

Если установлена опция "Командная переменная", пользователь может изменять значение связанной переменной двойным щелчком по графическому символу.

### A.18.4 Команды меню Файл

Меню "Файл" содержит команды, которые позволяют пользователю управлять документом.



Команда "Новый" меню "Файл" начинает редактирование нового документа. Количество документов в проекте не ограничивается ISaGRAF. Перед началом редактирования новой схемы ранее открытая схемы закрывается. Прожектор не может быть использован для редактирования нескольких схем одновременно. Тем не менее, несколько окон с Прожектором могут быть открыты одновременно для редактирования различных проектов.



Команда "Открыть" меню "Файл" позволяет пользователю закрыть текущий документ и начать редактировать другой документ текущего проекта. Новый выбранный документ замещает текущий в окне редактирования. Если Вы выбрали новый документ, кнопка "Удалить" может быть использована для существующего файла, для того, чтобы очистить директорию проекта. Если схема удаляется, то удаляются и иконки и битмапы связанные с ней.



Команда "**Сохранить**" меню "Файл" сохраняет текущий документ на диске. Если это новый документ, пользователь должен дать ему имя до сохранения. Наименование этого документа должно удовлетворять следующим правилам:

- Длина имени не может больше **8** символов
- Первый символ должен быть буквой
- Следующие должны быть буквами, цифрами или подчеркиком
- Заглавные и прописные буквы в имени воспринимаются одинаково

Команда "Сохранить как" меню "Файл" позволяет пользователю запоминать отредактированный документ под новым именем.

### **A.18.5 Замечание для пользователей ISaGRAF V3.2**

Прожектор может читать графику и списки временных диаграмм построенные средствами ISaGRAF V3.0 или V3.2. Такие файлы возникают в диалоге "**Открыть**", с описанием их источника. Файлы могут быть прочтены и свободно модифицированы Прожектором.

При открытии графики ISaGRAF V3.2, документ автоматически помечается как "Закрыто". Удалите опцию "Закрыто" из меню "**Файл**" если вы хотите изменять графику.

Если графика или список временных диаграмм ISaGRAF 3.2 открыт, Прожектор всегда предлагает сохранить его в родном формате Прожектора. Диалог "**Сохранить Как**" открывается, когда закрывается такой документ.

## A.19 Выгрузить

ISaGRAF поддерживает возможность выгрузки приложения, которое было сохранено в целевой задаче. Процедура выгрузки связывается с целевой задачей для того, чтобы загрузить встроенный упакованный код (ВУК) и, затем, восстанавливает загруженный проект в среде разработки.

Проект, работающий на соединенной целевой системе может быть выгружен если версия целевой задачи 3.22 или выше и если упакованный код был встроен в приложение. Встроенный исходный текст для загрузки - это дополнительная возможность. Смотрите следующий раздел, для того чтобы получить дальнейшую информацию о том, как должен быть сконфигурирован проект для выгрузки.

### A.19.1 Выгрузка проекта

Диалог «Выгрузить» запускается командой «Файл» менеджера проектов ISaGRAF. Команда Выгрузить не имеет отношения к существующему проекту в системе разработки. Выбранный в настоящее время проект в списке проектов не имеет отношения к механизму выгрузки. Чтобы выгрузить приложение работающее на целевой системе, Вы должны:

- 1-убедиться в том, что целевая задача правильно подключена
- 2-установить параметры связи в соответствии с типом связи
- 3-нажать кнопку Run

Выгрузка встроенного упакованного кода (ВУК) и его распаковка может занять несколько секунд. Сообщения в блоке диалога проинформируют Вас о том, что выгрузка закончена или сообщает об ошибке.

Имя, под которым создается проект ISaGRAF, считывается из целевой задачи. Если это имя уже используется для существующего проекта, Вы получите подсказку, предлагающую переписать проект или выбрать новое имя. Вы не можете отказаться от регистрации нового проекта, если он уже выгружен. Выгружаемый проект теперь готов и он может быть открыт.

#### ☰ **Возможные ошибки**

Следующие ошибки могут возникнуть при выгрузке проекта. О них Вы будете проинформированы.

- невозможно установить связь с целевой задачей
- целевая задача имеет более раннюю версию чем 3.22
- нет приложения в целевой задаче
- нет ВУК в целевой задаче

### A.19.2 Параметры связи

Кнопка **Установки** позволяет определить параметры связи системы разработки ISaGRAF и целевой задачи, используемые для Выгрузки. И вы должны убедиться

в то, что установленные параметры соответствуют параметрам подключенной целевой задачи до Выгрузки.

### A.19.3 Подготовка проекта к выгрузке

Вы должны проинформировать генератор кода, что упакованный исходный текст должен быть встроен в код приложения, если Вы хотите выгружать его в дальнейшем. Для этого нажмите на кнопку Выгрузить в блоке диалога Опции компилятора. Другой блок диалога позволяет определить встроенный исходный код. В этом случае будет встроено минимальное количество исходных файлов. Используйте другие блоки для того, чтобы встроить дополнительные файлы.

**Важное замечание:** библиотеки не загружаются с встроенным исходным кодом. Это касается функций, функциональных блоков, плат В/В и оборудования.

#### — **Дополнительные файлы**

В дополнение к минимальному требуемому исходному коду могут быть встроены следующие файлы. Включение этих файлов ведет к использованию дополнительной памяти.

##### *Дескриптор проекта*

Если не встроено то дескриптор проекта после выгрузки будет указывать только дату.

##### *Парольная защита*

Функция выгрузки не защищена паролем, если Вы хотите защитить выгруженный проект паролем Вы должны встроить систему парольной защиты в исходный код.

##### *Комментарии для несвязанных каналов В/В*

ISaGRAF дает возможность ввести комментарии для несвязанных каналов. Не выбирайте эту опцию, если Вы работаете только со связанными каналами.

##### *История модификаций*

Глобальная история изменений проекта.

##### *Файлы дневника*

Файл дневника для каждой программы содержит написанные пользователем замечания плюс историю сообщений компилятора относящуюся к программе. Встроенные файлы дневника могут потребовать много памяти в целевой системе.

##### *Списки переменных и временные диаграммы*

Эти файлы создаются во время отладки и содержат список переменных и временные диаграммы.

##### *Графики, иконки и битмапы*

Включает графики, иконки и битмапы, если они находятся в директории ISaGRAF.

### A.19.4 Как упакованный исходник запоминается в целевой задаче

Встроенный упакованный код (ВУК) запоминается в сгенерированном коде вместе с ресурсами. Сгенерированный ресурс называется «EVS» . Если выбран встроенный исходный код, то Вы не можете использовать это же имя для другого ресурса. Встроенный исходный код не налагает никаких ограничений на определение ресурсов. Встроенный исходный код никак не влияет на написанный пользователем файл определения ресурсов.

Смотрите документацию **Генератор кода ISaGRAF** для дальнейшей информации о ресурсах.

### **A.19.5 Требования к памяти целевой системы**

Встроенный упакованный код (ВУК) требует дополнительной памяти в целевой системе. По грубой оценке ВУК занимает в полтора раза больше места чем приложение. Это означает, что использование ВУК умножает размер требуемой памяти на 2.5.

Специальные ограничения могут возникнуть на целевых системах с сегментированной памятью. Так как ВУК запоминается как ресурс в сгенерированном коде, он должен быть сохранен в том же сегменте данных, что и код приложения.

### **A.19.6 О выгруженном проекте**

Выгруженный проект содержит все файлы и данные, которые могут потребоваться для перекомпиляции. В зависимости от выбранных перед компиляцией опций он может содержать дополнительные данные такие как дескриптор проекта и дневник программы.

Вы должны скомпилировать проект перед отладкой. Предупреждение: Так как ISaGRAF использует метку в скомпилированных данных, при запуске отладчика, Вы будете проинформированы если проекты имеют разные номера версий.

**Важное замечание:** библиотеки и функциональные блоки не выгружаются, поэтому Вы должны убедиться, что проект содержит соответствующие библиотеки до перекомпиляции выгруженного приложения.

### **A.19.7 Совместимость**

Выгрузка поддерживается в целевой задаче и системе разработки в версиях 3.22 и старше.

Нет ограничений на встроенный упакованный код (ВУК) в версиях от 3.03 до 3.21, так как ВУК запоминается как стандартный ресурс. Но встроенная информация не может быть выгружена в этом случае из-за того, что целевая задача не поддерживает требуемые коммуникационные функции.

## A.20 Использование средств Диагностики

“Средства диагностики” - это подмножество средств отладчика ISaGRAF. Оно позволяет конечному пользователю работать с predetermined набором переменных, для того, чтобы исследовать и управлять процессом. Отладчик ISaGRAF - очень мощное средство, которое включает функции высокого уровня. Средства Диагностики обеспечивают безопасный путь управления целевым приложением для последних запущенных операций или обслуживания. Средства Диагностики ISaGRAF запускаются непосредственно из группы ISaGRAF в Менеджере Программ двойным щелчком на следующей иконке:



Список существующих проектов отображается в окне диалога. Это позволяет пользователю запустить лимитированную версию отладчика ISaGRAF на существующем, уже загруженном приложении ISaGRAF. Нажатие кнопки “ПРИНЯТЬ” запускает лимитированный отладчик на выбранном проекте. Нажатие кнопки “Отказ” закрывает окно диалога. Команда “Установка” используется для установки связи между ISaGRAF Workbench и целевым PLC. Обратитесь к главе “Управление программами” этого руководства для получения большей информации по этой команде.

**Замечание:** Диагностические Средства ISaGRAF (лимитированный отладчик) не может быть использован для загрузки, остановки или обновления приложения, запущенного на целевом PLC. Если проект, выбранный в окне диалога Диагностических Средств, не тот же самый, что установлен и запущен на PLC, то ни одна операция не может быть выполнена.

Когда лимитированный отладчик ISaGRAF запущен и правильно подключен к целевому приложению, то доступны следующие команды:

- Просмотр списков переменных
- Просмотр графических документов с помощью Прожектора



## A.21 Использование симулятора ISaGRAF

Симулятор ядра ISaGRAF запускается вместе с отладчиком, когда выполняется команда “Симулировать” из меню “Отладка” в окне Управления Программ. Симулятор ядра - это законченная целевая система ISaGRAF, поддерживающая стандартные свойства ISaGRAF и все функции “С”, и блоки функций стандартной библиотеки, поставляемой CJ International. Платы В/В графически симулируются в окне. Любой тип платы В/В может быть симулирован. Платы, определенные как “Виртуальные платы” во время соединения В/В, также появляются в окне симуляции.

### A.21.1 Связи с отладчиком

Симулятор ядра поддерживает полную связь с отладчиком ISaGRAF, так что любые возможности отладчика могут быть использованы во время симуляции. Симулятор ядра всегда работает с текущим приложением ISaGRAF. Во время симуляции команды “Запустить”, “Остановить”, “Загрузить” или “Изменить” недоступны. Симулятор не может быть использован, если цель “Симуляция” не была выбрана в опциях компилятора перед построением целевых кодов. Закрытие окна симулятора означает, что окно отладчика (и любое окно ISaGRAF, открытое во время сессии отладки) будет также закрыто.

### A.21.2 Симуляция В/В

Платы В/В, появляющиеся в окне симулятора, подписываются по их имени и номеру разъема. Общаются любые стандартные типы В/В ISaGRAF (логический, аналоговый или сообщение). Каналы входных плат отображаются со специальными кнопками и полями. Каналы выходных плат отображаются с графическими индикаторами статуса и полями данных.




**Логические входы:** Логический вход представлен квадратной зеленой кнопкой. Номер канала отображается вместе с кнопкой. Входное значение - TRUE, когда кнопка нажата. Щелчок мышкой по кнопке изменяет соответствующее значение В/В. Используйте правую кнопку мышки для установки входа только тогда, когда кнопка нажата.





**Логические выходы:** Логический выход представлен маленьким кругом. Номер канала отображается вместе с В/В. Выходное значение - TRUE, когда графический символ высвечен..



**Аналоговые входы:** Канал аналогового входа - это простое числовое поле, где может быть введено значение соответствующего входа. Щелчок мышкой на прямоугольнике отобразит знак вставки. Тогда может быть введено новое значение для канала. Нет необходимости использовать клавишу **ENTER** после ввода. Аналоговые значения могут введены либо в десятичном, либо шестнадцатеричном виде. Используйте кнопки вверх/вниз для увеличения или уменьшения текущего значения.

 **Аналоговые выходы:** Канал аналогового выхода - это цифровое поле вывода. Выходное значение может быть отображено либо как десятичное, либо как шестнадцатеричное число. Пользователь не может выполнить ни одного действия на выходном канале.

 **Входы сообщений:** Канал входа сообщения - это простое текстовое поле, где вводится значение соответствующего входа. Щелчок мышкой на прямоугольнике отобразит знак вставки. Тогда может быть введено новое значение для канала. Нет необходимости использовать клавишу **ENTER** после ввода.

 **Выходы сообщений:** Канал выхода сообщения - это текстовое поле вывода. Пользователь не может выполнить ни одного действия на выходном канале.

### A.21.3 Компоненты библиотеки

Симулятор ISaGRAF полностью поддерживает стандартные преобразования, функции и блоки функций, поставляемые CJ International. Ниже дан список поддерживаемых объектов:

#### ⇒ **Функции преобразования:**

bcd, scale

#### ⇒ **Функции:**

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

#### ⇒ **Функциональные блоки:**

average, blink, cmp, ctd, ctu, ctud, derivate, f\_trig, hyster, integral, lim\_alrm, r\_trig, rs, sema, sr, stackint, tof, ton, tp

Преобразования, определенные пользователем, функции "С" и функциональные блоки обычно не интегрированы в симулятор ISaGRAF. Обычно такие объекты разрабатываются для использования ресурсов программного обеспечения и оборудования целевой системы. Такие ресурсы, как правило, недоступны в Windows системе. Симулятор ISaGRAF обеспечивает следующее стандартное поведение для любого, определенного пользователем, преобразования, функции или функционального блока:

- Когда новое преобразование совершается симулятором, оно замещается "null" преобразованием. Это означает, что физическое значение аналоговых переменных всегда равно электрическому значению (как введено или отображено на панели Симулятора).
- Если новая функция "С" или функциональный блок запущен симулятором, то не производится ни одной операции. Результирующее значение не устанавливается.

#### A.21.4 Опции

Команды меню "**Опции**" позволяют пользователю управлять дисплеем В/В в панели симуляции. Пользователь может установить или удалить эти опции в любое время в течении отладки.

- ⇒ Когда установлена опция "**Цветной дисплей**", каналы В/В отображаются как цветные bitmap картинки. Если цвета не различаются на некоторых LCD дисплеях, то пользователь должен убрать эту опцию, что бы получить чистую черно-белую графику ввода-вывода для каналов В/В.
- ⇒ Когда установлена опция "**Имена переменных**", то позади любого канала В/В отображается полоска с именем подключенной переменной В/В. Удаление этой опции позволяет пользователю уменьшить размер панели симулятора.
- ⇒ Когда установлена опция "**Шестнадцатеричные значения**", любой канал аналогового В/В отображается или вводится в шестнадцатеричном формате.
- ⇒ Когда установлена опция "**Всегда наверху**", окно симулятора всегда видно, даже если место ввода находится в другом окне.

#### A.21.5 Сохранение и восстановление состояний входов

Используя симулятор ISaGRAF, входными каналами можно управлять, действуя на кнопки и управление редактирования панели симулятора. Вы можете в любое время использовать следующие команды меню "**Инструменты**" чтобы сохранять и восстанавливать состояние всех входных каналов:

##### **Загрузить входную схему**

Присвоить входным каналам значения сохраненные в файле который был создан на диске с помощью команды "Сохранить входную схему".

##### **Сохранить входную схему**

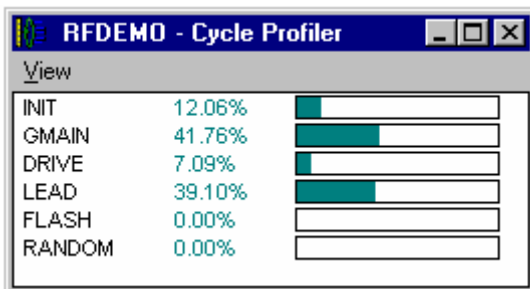
Сохранить состояния входных каналов в файле, так чтобы они могли быть восстановлены позднее с помощью команды "Загрузить входную схему". Файл сохраняется в директории проекта и таким образом сохраняется с другими файлами проекта с помощью архиватора ISaGRAF.

Замечание: Только именованные входные каналы (имеющие присоединенные переменные) сохраняются на диске.

#### A.21.6 Профилер цикла

Профилер цикла ISaGRAF – это мощное диагностическое средство, которое показывает, как время цикла распределяется между различными программами, функциями и функциональными блоками приложения. Это средство очень полезно для быстрой диагностики производительности приложения и приводит программиста к той части кода, которая требует оптимизации.

Профилер цикла запускается командой "Инструменты / Профилер цикла" в меню окна симулятора ISaGRAF. Он показывает, для каждой программы, функции или функционального блока, процент потраченного времени цикла:



Если установлена опция "Вид / Средний", представленная информация – это средние проценты рассчитанные с момента запуска приложения, или с последнего запуска команды "Вид / Сброс".

Если не установлена опция "Вид / Средний", представленная информация – это измерения сделанные в течении последнего цикла. Вы можете также использовать эту возможность, когда приложение находится в режиме "Пошаговый" чтобы получить измерения зависящие от контекста приложения.

Используйте команду "Вид / Копировать" чтобы копировать имена программ и проценты в клипборд Windows в ASCII формате. Затем данные могут быть вставлены в текстовый документ или электронную таблицу.

### **Важные замечания:**

Это неточные измерения. Вычисление процентов основано на подсчете TIC инструкций, принимая во внимание различные времена выполнения инструкций. Вычисление не включает время потраченное на "C" функции и функциональные блоки.

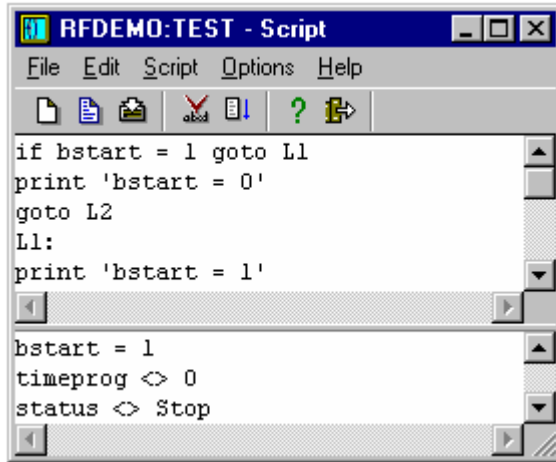
Величина появляющаяся для функции или функционального блока – это сумма всех "времен вызова" из прикладных программ в одном цикле.

Вычисление времени основано на TIC коде и не обеспечивает надежной информации, если действительный код приложения сгенерирован на языке "C" и построен с использованием компилятора "C".

## **A.21.7 Скрипты симуляции**

ISaGRAF симулятор включает средство для построения и запуска скриптов симуляции. Скрипт описывается на простом языке типа ST, и используется для тестирования с помощью симулятора ISaGRAF.

Редактор скрипта симуляции запускается командой "Инструменты / Скрипты симуляции" окна Симулятора. Ниже дана рамка редактора скрипта:



Верхнее окно – это текстовый редактор, в котором вводятся инструкции скрипта. Оно используется также, как и другие текстовые редакторы ISaGRAF и включает возможности верхнего уровня, такие как выбор мышкой символа переменной. Вы можете использовать команды меню "Опции" для установки ширины табуляции и выбора шрифта.

Нижнее окно показывает все сообщения при работе скрипта. Разделительная линия между окнами может быть свободно передвинута чтобы изменить размер окон. Выходное окна может быть скрыто во время редактирования скрипта, но автоматически открывается каждый раз когда запускается скрипт.

## ⊞ **Редактирование скриптов**

Используйте команды меню "Файл" для управления файлами скриптов:

**Новый** ..... создает новый скрипт без заголовка

**Открыть** ..... загружает существующий скрипт из файла

**Сохранить** ..... сохраняет текст скрипта и содержание выходного окна на диске директории проекта

**Сохранить как...** сохраняет скрипт под другим именем

Для каждого скрипта в директории проекта ISaGRAF создаются два файла:

<имяскрипта>.SCC..... текст скрипта (инструкции)

<имяскрипта>.SCO ..... содержание выходного окна

где <имяскрипта> - имя скрипта. Оба файла – стандартные текстовые файлы и могут быть открыты любым текстовым редактором.



Во время редактирования скрипта, вы можете использовать команду "**Редактор/ Вставить символ**" чтобы выбрать объявленное имя переменной и ввести его в позиции курсора.

## ⊞ **Запуск скриптов**

Перед запуском скрипт должен быть проверен и скомпилирован. Если нужно, синтаксис проверяется автоматически по команде "Запустить". Используйте следующие команды меню "**Скрипт**":



**Проверить** ..... проверить синтаксис и компилировать скрипт

**Запустить скрипт** ... запустить выполнение текущего скрипта

В случае нового скрипта без заголовка он должен быть сохранен (и имя должно быть введено) перед проверкой. В случае именованного скрипта, скрипт автоматически сохраняется на диск перед проверкой синтаксиса.

Если скрипт запущен, его содержание не может быть изменено. Когда достигается конец скрипта, появляется сообщение. Вы можете также прервать работу скрипта используя следующую команду меню "**Скрипт**":



**Прервать скрипт** .... завершает работу скрипта

Исполнение скрипта происходит между целевыми циклами. В случае бесконечного витка в программном цикле, симулятор ISaGRAF обеспечивает, что этот виток всегда прерывается так, что циклы ISaGRAF выполняются и другие приложения ISaGRAF не блокируются. Интерпретатор скрипта ISaGRAF решает прервать скрипт, если та же "метка" встречается более одного раза в одном целевом цикле. Исполнение скрипта может быть прервано инструкциями "Cycle" или "Wait".

## ▬ **Язык описания скрипта**

Язык описания скрипта – очень простой текстовый язык похожий на ST, но где каждая инструкция вводится в отдельной строке, и не требует точки с запятой в конце. Используйте следующую кнопку панели инструментов для вызова списка имеющихся инструкций и ввода ключевого слова в позиции курсора:



вставить инструкцию (ключевое слово и помощь как комментарий)

Существуют различные типы инструкций. Во-первых присвоение переменной:  
:= .....присвоение

Другие инструкции позволяют выводить сообщения в окно вывода:

**Print**.....выводит текстовую строку или значение переменной

**PrintTime** ..... выводит текущую временную отметку

Другие инструкции используются для синхронизации инструкций скрипта с циклом ISaGRAF:

**Cycle**..... пусть симулятор ISaGRAF выполнит один цикл

**Wait**.....ждать в течении определенного времени

Другие инструкции используются для управления потоком в скрипте:

**Labels** ..... может быть помещена в любое место скрипта

**Goto** .....безусловный переход на метку

**If goto**..... условный переход на метку

**End**.....заканчивается скрипт

Язык скрипта не различает заглавных и прописных букв. Комментарии могут быть введены в конце любой строки. Комментарии могут быть написаны либо в

соответствии с соглашениями ST (между символами "(" и "\*"), или с предшествующим символом ";".

## " := " Присвоение

**Значение:** Присвоить значение переменной ISaGRAF. Она может быть внутренней переменной, входным каналом или выходным каналом.

**Синтаксис:** `<varname> := <constant_expression>`  
`<varname> = <constant_expression>`

**Аргументы:** `<varname>` символ объявленной переменной или непосредственно представленная переменная В/В использующая соглашения "%".

`<constant_expression>` константа, которая соответствует типу переменной. Для булевских, "0" и "1" могут использоваться "FALSE" и "TRUE". Для таймеров, префикс "T#" или "TIME#" может быть опущен.

**Замечание:** Входная переменная устанавливаемая скриптом не требует блокировки. Чертеж соответствующего входного канала изменяется, когда входная переменная изменяется скриптом.

**Предупреждение:** не изменяйте входную или выходную переменную присоединенную к преобразованию, так как исполнение скрипта не поддерживает функции и таблицы преобразования.

**Пример:**

```
MyBooVar := 1           (* то же, что и TRUE *)
MyIntVar := 1234
MyRealVar := 1.2345
MyMsgVar := 'Hello'
MyTmrVar := t#12s
```

## Print

**Значение:** Пишет строку или значение переменной в окно вывода. Текст выводится как одна новая строка в конце уже написанного текста в окне вывода.

**Синтаксис:** `Print '<text>'`  
`Print <varname>`

**Аргументы:** `<text>` любая строка в одиночных кавычках

`<varname>` символ объявленной переменной или непосредственно представленная переменная В/В использующая соглашения "%".

**Замечание:** Выходная переменная всегда форматируется в соответствии с соглашениями IEC.

**Пример:**  
`Print 'Hello'  
Print MyBooVar`

**Выход:**  
Hello  
MyBoovar = TRUE

## PrintTime

**Значение:** Пишет текущую временную отметку в выходное окно. Текст выводится как одна новая строка в конце уже написанного текста в окне вывода.

**Синтаксис:** `PrintTime`

**Замечание:** Временная отметка форматируется в соответствии с текущими установками Windows

**Пример:**  
`Print 'Time now is:'  
PrintTime`

**Выход:**  
Time now is:  
15:45:22

## Cycle

**Значение:** Подвешивает исполнение скрипта до выполнения следующего цикла ISaGRAF.

**Синтаксис:** `Cycle`

**Замечание:** Инструкции скрипта исполняются в начале цикла ISaGRAF. Если симулятор в пошаговом режиме, за инструкцией "cycle" следует сразу цикл. Следующие инструкции цикла будут выполнены на следующей команде отладчика "Выполнить один цикл".

**Пример:**  
`(* программа ISaGRAF копирует A в B *)  
A := 0  
Cycle  
Print B  
A := 1  
Print B (* цикл не выполнен/B не установлено в 1 *)  
Cycle  
Print B`



**Выход:** B = 0  
B = 0  
B = 1

## Wait

**Значение:** Подвешивает исполнение цикла до истечения задержки

**Синтаксис:** `Wait <delay>`

**Аргументы:** `<delay>` задержка выраженная в соответствии с соглашениями IEC для временных констант. Префиксы "T#" или "TIME#" могут быть опущены. Величина задержки должна быть между 10 миллисекундами и 1 часом.

**Замечание:** Точность инструкции "Wait" зависит от системы Windows. Также, задержка должна рассматриваться с точностью плюс минус один цикл ISaGRAF. Если инструкция "Wait" достигнута, циклы ISaGRAF выполняются до тех пор пока не истечет задержка и перед продолжением выполнения скрипта.

**Пример:**  
`PrintTime`  
`Wait 2s`  
`PrintTime`

**Выход:** 15:45:27  
15:45:29

## Метки

**Значение:** Метки могут быть помещены в любое место скрипта. Они используются, как назначение для инструкции "Goto" и позволяют управлять потоком инструкций скрипта.

**Синтаксис:** `<labelname>:`

**Аргументы:** `<labelname>` уникальное имя в соответствии с соглашениями ISaGRAF: ограничено 16 символами, начинается с буквы, далее идут буквы, цифры или подчеркик. Если определена, то за именем метки должен следовать символ ":".

**Замечание:** Нельзя помещать инструкции в строке, где определена метка. Имя метки не должно совпадать с именем объявленной переменной ISaGRAF

**Пример:** (\* Пример скрипта с бесконечным циклом \*)  
`loop:`

```
PrintTime
Wait 1s
Goto loop
```

## Goto

**Значение:** Безусловный переход на метку

**Синтаксис:** `Goto <labelname>`

**Аргументы:** `<labelname>` имя метки определенное в скрипте

**Замечание:** Разрешены переходы назад. В случае бесконечного цикла, выполнение скрипта автоматически прерывается на каждом витке, для того чтобы сохранить выполнение циклов ISaGRAF.

**Пример:**

```
Print 'Before Jump'
Goto MyLabel
Print 'Within Jump' (*никогда не выполняется *)
MyLabel:
Print 'After Jump'
```

**Выход:**

```
Before Jump
After Jump
```

## If Goto

**Значение:** Условный переход на метку. Условие – это либо сравнение двух переменных ISaGRAF, либо сравнение переменной и константы.

**Синтаксис:** `If <var1> test <var2> Goto <labelname>`  
`If <var1> test <constant_expr> Goto <labelname>`

Имеются **тесты** сравнения:

<code>=</code>	true если члены имеют одно значение
<code>&lt;&gt;</code>	true если члены имеют разные значения
<code>&lt;</code>	true если первый член меньше второго
<code>&lt;=</code>	true если первый член меньше или равен второму
<code>&gt;</code>	true если первый член больше второго
<code>&gt;=</code>	true если первый член больше или равен второму

**Аргументы:** `<var1>` `<var2>` символы объявленных переменных или непосредственно представленные переменные В/В использующие соглашения "%".

`<constant_expr>` константа, которая соответствует типу переменной. Для булевских, "0" и "1" могут использоваться "FALSE"

и "TRUE". Для таймеров, префикс "T#" или "TIME#" может быть опущен.

`<labelname>` имя метки определенной в скрипте.

**Замечание:** Разрешены переходы назад. В случае бесконечного цикла, выполнение скрипта автоматически прерывается на каждом витке, для того чтобы сохранить выполнение циклов ISaGRAF.

**Пример:**

```
(* Этот виток работает до тех пор пока MyVar не равно TRUE *)
Loop:
If MyVar = TRUE Goto TheEnd
Print MyVar
Goto Loop
TheEnd:
```

## End

**Значение:** Заканчивает скрипт

**Синтаксис:** **End**

**Замечание:** Инструкция "End" должна быть в последней строке скрипта

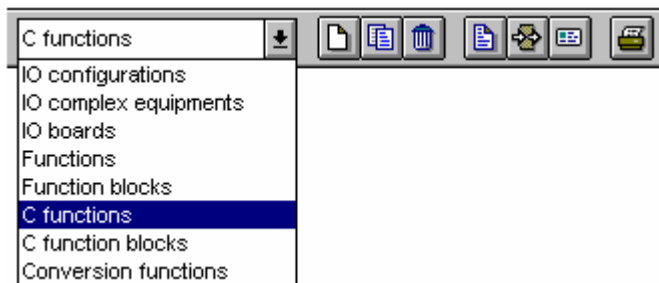
**Пример:**

```
(* Этот виток работает до тех пор пока MyVar не равно TRUE *)
Loop:
If MyVar = FALSE Goto Continue
End
Continue:
Print MyVar
Goto Loop
```

## A.22 Использование менеджера библиотек

Библиотеки ISaGRAF обеспечивают стандартный интерфейс между автоматической разработкой и возможностями программного обеспечения или оборудования целевой системы ISaGRAF. Существует по одной библиотеке для каждого типа интерфейса. Менеджер Библиотек ISaGRAF Workbench предназначен для разработчика оборудования или программного обеспечения. Разработчик использует менеджер библиотек для описания интерфейса программирования ISaGRAF с объектами, которые он создает.

Менеджер Библиотек ISaGRAF Workbench показывает элементы одной из библиотек ISaGRAF. В левой области окна находится список элементов выбранной библиотеки. В правой области находятся технические замечания (руководство пользователя) по элементу, в настоящий момент выбранному в списке элементов. Меню менеджера библиотек содержат команды для создания, определения или изменения элементов активной библиотеки. Команда **“Файл / Другая библиотека”** разрешает выбор одной из библиотек ISaGRAF. Окно с левой стороны от меню средств тоже может быть использовано для выбора библиотеки:



### A.22.1 Управление элементами библиотеки

Используйте команды меню **“Файл”** для создания элементов и работайте с существующими элементами в открытой библиотеке.



#### **Создание нового элемента**

Команда **“Новый”** из меню **“Файл”** создает новый элемент в выбранной библиотеке. Имя нового элемента вводится на основе следующих правил:

- Длина имени не должна превышать **8** символов
- Первый символ должен быть **буквой**
- Следующие должны быть **буквами, цифрами** или символом подчеркивания
- Верхний и нижний регистры в наименовании не различаются.

Текстовые комментарии связываются с каждым элементом библиотеки. Этот комментарий вводится во время создания элемента. Когда создается новый элемент, то должно быть введено следующее:

- его определение для конфигурации В/В,
- его параметры для платы В/В,

- его пользовательский интерфейс для функции или функционального блока. Когда создается преобразование “С”, функция “С” или функциональный блок “С”, то полный фрейм его исходного кода генерируется автоматически.

## ▬ **Работа с существующими элементами**

Команда **“Файл / Переименовать”** позволяет пользователю изменить имя или комментарий элемента, выбранного из списка элементов. Команда **“Файл / Копировать”** позволяет пользователю копировать элемент, высвеченный в активной библиотеке, на другой элемент в той же самой библиотеке. Если элемент назначения уже существует, то все его содержание будет перезаписано. Если элемент назначения не существует, то он будет автоматически создан. Команда **“Файл / Удалить”** удаляет текущий выбранный элемент из активной библиотеки. Следующие компоненты элемента управляются командами **“Переименовать”, “Копировать”** и **“Удалить”**:

- техническое примечание
- полное определение для конфигурации В/В
- параметры для платы В/В или сложного оборудования
- определение интерфейса для функции или функционального блока
- исходные коды для функции и функционального блока, написанного на языке IEC
- исходные коды для преобразования С, функции или функционального блока



Если элемент - это преобразование “С”, функция “С” или функциональный блок “С”, то его имя автоматически не обновляется командами **“Переименовать”** или **“Копировать”** в подключенном исходном коде.



Если элемент - это функция, написанная на языке IEC, то имя возвращаемого параметра не изменяется командами **“Переименовать”** или **“Копировать”**.

## ▬ **Установка парольной защиты**

Команда **“Файл / Установить пароль”** позволяет пользователю определить парольную защиту для выбранного элемента в открытой библиотеке. Обратитесь к разделу **“Парольная защита”**, в конце первой части этого руководства для получения дальнейшей информации об уровнях паролей и защите данных. Пароли относятся только к выбранному элементу. Они не имеют никакого влияния на другие элементы в библиотеке ISaGRAF.

## ▬ **Компилирование функций и функциональных блоков**

Когда выбрана библиотека функций или функциональных блоков, написанная на языке IEC, команда **“Проверить (Компилировать)”** меню **“Файл”** используется для проверки синтаксиса выбранного элемента и создания его объектного кода. Функции и блоки, написанные на языке IEC, должны быть скомпилированы без ошибок, прежде чем они могут быть использованы в проектах ISaGRAF. Эта команда не эффективна, если выбрана другая библиотека.



### **Технические замечания**

Команда **“Редактор/ Технические замечания”** позволяет пользователю определить стандартный текстовый формат для всех элементов текущей

выбранной библиотеки. Во время редактирования технического замечания нового элемента, этот формат используется как основной скелет. Это позволяет пользователю оптимизировать редактирование технического замечания.

Команда "**Инструменты / Стандартный формат замечания**" позволяет пользователю определить стандартный текстовый формат для всех элементов текущей выбранной библиотеки. Во время редактирования технического замечания нового элемента, этот формат используется как основной скелет. Это позволяет пользователю оптимизировать редактирование технического замечания.



## **Параметры**

Параметры элемента описывают **интерфейс** между компьютерными операциями, обеспечиваемыми элементом, и использованием элемента в приложении ISaGRAF. Параметры имеют различное значение для каждого типа элемента библиотеки.

Параметры конфигурации В/В определяют полный набор плат В/В конфигурации, и имена переменных по умолчанию, используемых для каналов В/В. Параметры платы В/В или сложного оборудования определяют физическую или логическую конфигурацию платы. Параметры функции или функционального блока определяют интерфейс элемента, в соответствии с соглашениями вызова функций ST языка. Для функции преобразования не существует параметров, так как она использует стандартный предопределенный интерфейс.



## **Исходные коды C**

ISaGRAF Workbench позволяет программисту управлять исходным кодом библиотечного преобразования, функции или функционального блока. Исходный код функции или блока, написанный на языке IEC, - это текст или диаграмма, описанная языком, прикрепленным к функции. Исходные коды "C" компонентов ("C" функции, функциональные блоки "C" и функции преобразования) разделяются на два отдельных файла: **исходный код C заголовка**, который содержит именно определение **интерфейса**, в соответствии с определением параметров элемента, и **файл исходного кода C**, который содержит реализацию работы элемента.

ISaGRAF Workbench генерирует файл исходного кода, когда создается новый библиотечный элемент. Он также создает и обновляет исходный заголовок, основанный на определении параметров. Программист может использовать текстовый редактор ISaGRAF для завершения файла исходного кода.



## **Архивация элементов библиотеки**

Команда "**Инструменты / Архив**" запускает менеджер архивов ISaGRAF чтобы сохранить или восстановить элементы библиотеки. Вы должны выбрать библиотеку перед запуском "**Команды архив**". Менеджер архивов показывает элементы только одной библиотеки за раз.

## A.22.2 Конфигурация В/В

Библиотека конфигурации В/В ISaGRAF обеспечивает легкий путь для инициализации новых проектов ISaGRAF с предопределенной конфигурацией В/В. Конфигурация В/В определяет:

- набор плат В/В
- значения по умолчанию для параметров плат В/В
- имена по умолчанию для каналов В/В

Когда создается новый проект ISaGRAF с библиотечной конфигурацией В/В, соответствующее соединение В/В автоматически устанавливается, и переменные В/В, соответствующие именам каналов, автоматически объявляются в словаре проекта.



Определение конфигурации В/В делается при помощи средств Соединения В/В ISaGRAF (то же самое средство используется внутри проекта). Обратитесь к разделу "Соединение В/В" в руководстве для получения дальнейшей информации по тому, как использовать это средство. Когда вставляется новая плата В/В в конфигурацию, все каналы новой платы объявляются со стандартными именами по умолчанию. Стандартное имя по умолчанию канала В/В имеет следующий формат:

**<направление><тип><номер\_разъема>\_<номер\_канала>**

Первый символ обозначает направление канала В/В:

"I" ..... канал ввода  
"O" ..... канал вывода

Второй символ обозначает тип канал В/В:

"X" ..... логический  
"D" ..... аналоговый  
"M" ..... сообщение

Ниже даны примеры стандартных имен каналов В/В:

**IX0\_7** ..... логический вход - плата №0 - канал №7  
**QD2\_4** ..... целочисленный выход - плата №2 - канал №4

Команда "Соединить каналы В/В" Редактора Соединений В/В используется для изменения имени по умолчанию, присоединенного к каналу В/В.

## A.22.3 Сложное оборудование В/В

Все каналы одиночной платы имеют тот же самый тип (логический, аналоговый или сообщение) и направление (ввод или вывод). Сложное оборудование В/В представляет устройство В/В с каналами различных типов или направлений. Сложное оборудование В/В представляется как список одиночных плат В/В. Оно использует только один разъем в списке шасси соединений В/В.



Для определения сложного оборудования В/В пользователь должен определить список одиночных плат, которые определяют оборудование В/В. Он также должен

ввести детальные параметры каждой одиночной платы. Список одиночных плат В/В вводится при помощи окна диалога.

Нажатие кнопки **“Добавить”** позволяет пользователю добавить одиночную плату в конец текущего списка. Кнопка **“Вставить”** используется для вставки новой одиночной платы перед текущей выбранной платой в списке. Кнопка **“Удалить”** удаляет выбранную одиночную плату из списка. Кнопки **“Переименовать”** и **“Параметры”** используются для изменения имени и параметров выбранной одиночной платы. Обратитесь к соответствующему разделу для полного объяснения параметров одиночной платы. Сложное оборудование В/В может группировать до **16** одиночных плат В/В. Имя одиночной платы (внутри оборудования В/В) не может превышать **8** символов.

## A.22.4 Плата В/В

Библиотека плат В/В ISaGRAF определяет стандартный интерфейс между переменными приложения и целевым оборудованием. Во время описания приложения все переменные В/В подключаются к каналам целевой платы В/В. Плата В/В ISaGRAF определяется по имени и по **“ключевому коду OEM”** (**“OEM key code”**), который определяет его поставщика. Другие параметры платы В/В описывают топологию платы В/В (количество каналов, направление канала и тип), и конфигурацию, оборудования или программную.



### **Параметры платы В/В**

Существует два различных типа параметров для платы В/В: общие параметры, которые определяются для любой платы библиотеки ISaGRAF, и OEM параметры, которые специфичны для реализации платы, обеспечиваемые поставщиком оборудования. Общие параметры вводятся в верхней части окна определения параметров платы В/В. Эти параметры (плюс имя платы В/В) идентифицируют стандартный интерфейс платы В/В ISaGRAF.

**“Ключевой код OEM”** - простое число, которое определяет **поставщика оборудования**. Все платы, определенные одним поставщиком, должны иметь один и тот же ключевой код OEM. Ключевой код OEM - это **16 битное беззнаковое слово**, введенное в **шестнадцатеричном** формате. Резервированный ключевой код OEM для CJ International равен **“1”**.

Главные параметры определяют топологию платы В/В. **Число каналов** определяют число доступных каналов на плате. **Тип** платы - это тип переменных, которые могут быть подключены к каналам платы. **Направление** определяет какие переменные подключены к плате - **входные** или **выходные**.

**Замечание:** Переменные В/В различных типов или направлений не могут быть сгруппированы на одной и той же плате В/В ISaGRAF. Это свойство требует сложного оборудования В/В.



### **Параметры OEM**

Параметры OEM вводятся в нижней части окна определения параметров платы В/В. Эти параметры определяются поставщиками плат В/В и являются специфичными для платы. Существует максимум 16 OEM параметров для платы. Плата может не иметь OEM параметры. Менеджер библиотек ISaGRAF позволяет поставщику оборудования определить идентификацию и формат каждого параметра, и метод их ввода.



Квадрат слева содержит список OEM параметров. Каждый параметр идентифицируется **именем** и логическим **номером**, с **0** до **15**. Площадь справа содержит детальное описание параметра, выбранного слева. Параметр выбирается в списке для доступа к его полному описанию. Нажатие кнопки **“Очистить”** сбрасывает описание параметра и удаляет его из списка параметров.

Предупреждение: отмена этой команды невозможна.

Имя параметра используется для идентификации соответствующего входного поля во время соединения платы В/В, если поле должно быть определено автоматическим оператором. Имя параметра должно удовлетворять следующим правилам:

- максимальная длина имени **16** символов
- первый символ должен быть **буквой**
- следующие символы должны быть **буквами, цифрами** или символом ‘\_’.

Тип параметра определяет **внутренний формат** параметра и его **входной параметр** во время соединения В/В приложения. Ниже дан список возможных внутренних форматов:

**word** ..... беззнаковое 16 битное слово  
**long** ..... беззнаковое 32 битное слово  
**word hexa** ..... беззнаковое 16 битное слово  
**long hexa** ..... беззнаковое 32 битное слово  
**boolean** ..... беззнаковое 16 битное слово (только младший бит используется)  
**character** ..... беззнаковое 16 битное слово (только младший байт используется)  
**string** ..... массив из 16 байт, содержащий строку, оканчивающуюся нулем  
**float** ..... простой точности 32 битное плавающее значение

Ниже даны возможные входные форматы:

**word** ..... беззнаковое десятичное слово  
**long** ..... беззнаковое длинное слово  
**word hexa** ..... беззнаковое шестнадцатеричное слово  
**long hexa** ..... беззнаковое шестнадцатеричное длинное слово  
**boolean** ..... “true” или “false”  
**character** ..... один символ  
**string** ..... строка в кодах ascii (максимум 15 символов)  
**float** ..... простой точности плавающее значение

Окно **“доступ”** используется для определения того, как конечный пользователь может получить доступ к параметру. Если установлена опция **“Пользователь”**, то параметр показывается как входное поле во время соединения платы В/В. Значение по умолчанию OEM параметра используется как начальное для редактирования параметра. Если установлена опция **“Скрытый”**, то параметр является константой и не появляется в окне соединения платы В/В. Значение по умолчанию OEM параметра определяет значение параметра константы. Опция **“Только на чтение”** индицирует, что параметр видим для пользователя, но не может быть изменен. Его значение по умолчанию используется как константа.

## A.22.5 Функции и блоки, написанные на языке IEC

ISaGRAF управляет библиотекой функций и функциональных блоков, написанных на языке IEC. Доступные языки для описания таких функций или блоков - это **FBD** (Function Block Diagram), **LD** (Ladder Diagram), **ST** (Structured Text), **IL** (Instruction List). Заметьте, что языки LD и FBD могут быть смешаны в одной и той же диаграмме. Язык **SFC** (Sequential Function Chart) не может быть использован для описания функции или блока в библиотеке. Язык, прикрепленный к элементу библиотеки, выбирается когда создается функция, и не может быть изменен позже.

### **Компиляция**

Функции и блоки, определенные в библиотеке, должны быть скомпилированы (проверены) перед тем, как они могут быть использованы в проекте ISaGRAF. Более ничего не должно быть изменено со стороны Библиотеки, касающееся функций и блоков. Элементы библиотеки автоматически появляются в окне меню выбора, когда используется графический редактор LD/FBD внутри проекта.



Функция, определенная в библиотеке, может вызывать другую библиотечную функцию. Однако система ISaGRAF не поддерживает рекурсивность в вызове функций. Функциональный блок, написанный на языке IEC, не может вызывать другие функциональные блоки (ни на IEC, ни на языке "C").



### **Ввод исходных кодов**

Исходный код библиотечной функции или функционального блока вводится, используя стандартные средства ISaGRAF: графический редактор для LD или FBD программ, текстовый редактор для ST или IL программ. Обратитесь к соответствующему разделу в этом руководстве для получения дальнейшей информации об этих средствах. Генератор Кодов ISaGRAF может быть напрямую вызван из графического или текстового окна редактирования для завершения исходного кода библиотечной функции или блока.



### **Словарь локальных переменных**

Библиотечная функция или функциональный блок может иметь локальные переменные и локально определенные слова. Для доступа к декларированию переменной пользователь должен выполнять команды "Словарь" из меню "Файл" в окне редактора, во время редактирования исходного кода функции.



Библиотечная функция или функциональный блок не имеют доступа к глобальной переменной или экземпляру функционального блока. Локальные переменные функции должны быть инициализированы в теле функции.

Локальные переменные функционального блока, написанные на языке IEC, копируются (экземпляр) каждый раз, когда блок используется в проекте. Локальные переменные экземпляра сохраняют свои значения от вызова к вызову.



### **Определение интерфейса**

Функции или функциональные блоки могут иметь до 32 параметров (входов или выходов). Функция всегда имеет один (и только один) возвращаемый параметр,

который должен иметь то же самое имя, что и функция, для соответствия соглашениям языка ST. Следующее окно диалога используется для описания параметров функции или блока:

Список в верхней левой части окна показывает параметры, в порядке вызывающей модели: сначала параметры вызова, потом параметры возврата. Нижняя часть окна показывает детальное описание параметра, в данный момент выбранного в списке. Любой из типов данных ISaGRAF может быть использован для параметра. Параметры возврата должны быть расположены в списке после параметров вызова. Наименование параметров должно соответствовать следующим правилам:

- длина имени не должна превышать 16 символов
- первый символ должен быть буквой
- следующие должны быть буквами, цифрами или символом подчеркивания
- верхний и нижний регистры в наименовании не различаются

Команда **“Вставить”** используется для вставки нового параметра перед выбранным параметром. Команда **“Удалить”** используется для удаления выбранного параметра. Команда **“Упорядочить”** автоматически переставляет (сортирует) параметры так, что параметр возврата ставится в конец списка.

## A.22.6 “С” функции и функциональные блоки

“С” функции и функциональные блоки - это компьютерные функции, вызываемые из приложения автоматизации, в соответствии с интерфейсом вызова функций языка ST.

Функции являются синхронными процессами. Целевое приложение ISaGRAF приостанавливается во время выполнения функции. Функциональные блоки объединяют операции и статические скрытые данные. Например, функция “счетчик” представляет операцию счета, а также результат счета. Функции и функциональные блоки могут быть использованы для пополнения возможностей стандартного языка автоматизации, или для доступа к системным ресурсам.



Окно определения параметров используется для определения имени и типа каждого вызываемого или возвращаемого параметра функции или функционального блока. Меню команд **“Редактировать”** используется для определения параметров выбранной функции или функционального блока. Функция может иметь до 31 вызываемого параметра и всегда один возвращаемый параметр. Функциональный блок может иметь до 32 параметров с любым соотношением вызываемых и возвращаемых параметров. Ниже дано соответствие между типами ISaGRAF и типами “С”:

<b>BOOLEAN</b>	unsigned long	беззнаковое 32 битное слово: 1=true / 0=false
<b>ANALOG</b>	long	знаковое целое 32 битное слово
<b>REAL</b>	float	простой точности плавающее значение
<b>TIMER</b>	unsigned long	беззнаковое целое 32 битное слово (блок - 1 ms)
<b>MESSAGE</b>	char *	строка символов

Когда содержание сообщения передается в “С” функцию или функциональный блок, оно не может содержать нулевые символы. Строка, передаваемая в “С” код, заканчивается нулем.

Обратитесь к ISaGRAF Target User's Guide для получения дальнейшей информации по тому, как управлять исходным кодом "С" функции или функционального блока, и как интегрировать новый элемент в целевую систему ISaGRAF.

## **A.22.7 Функции преобразования**

Функция преобразования - это "С" функция, вызываемая менеджером В/В ISaGRAF каждый раз, когда аналоговая переменная, использующая это преобразование, является входом или выходом проекта.

Функция создает отношение между электрическим значением переменной (считанное на входном датчике или посланное на выходное устройство) и его физическим значением (используемое в выражениях приложения). Следовательно, функция делится на две части: входное преобразование и выходное преобразование. Менеджер библиотек ISaGRAF позволяет пользователю управлять исходным кодом "С" функции преобразования.

Преобразование может быть использовано для целочисленных и вещественных аналоговых переменных. Это значит, что интерфейс функции преобразования всегда определяется плавающими значениями. Интерфейс одинаков для любой функции преобразования. Определение "С" этого интерфейса сделано в файле определения **"TACN0DEF.H"**.

Обратитесь к ISaGRAF Target User's Guide для получения дальнейшей информации по тому, как управлять исходным кодом "С" функции преобразования, и как интегрировать новый элемент в целевую систему ISaGRAF.

## A.23 Использование утилиты архивирования (Архив)

Утилита архивирования ISaGRAF позволяет пользователю сохранять проекты ISaGRAF и библиотеки на дискетах или в каталоге дублирования. Менеджер Архивов ISaGRAF – это диалог, который может быть вызван из Менеджера Проектов или Менеджера Библиотек ISaGRAF.



Предполагается, что для создания и содержания надежных архивов, будут использоваться следующие положения::

- Записывайте имя и описание сохраненного объекта на наклейку диска
- Не сохраняйте проекты и библиотеки на одной и той же дискете
- Не сохраняйте разные проекты на одной и той же дискете

### A.23.1 Вызов менеджера архивов

Диалог "Архив" может быть вызван из меню "**Инструменты / Архив**" окна Управления Проектами, чтобы сохранить или восстановить проект или общие данные.

Диалог "Архив" может также быть запущен командой "**Инструменты / Архив**" Менеджера Библиотек ISaGRAF, чтобы сохранить или восстановить элементы библиотеки выбранной в настоящее время в окне Менеджера Библиотек.

#### ▬ **Проекты**

Проект всегда сохраняется в его полной форме. Все компоненты проекта (исходные программные файлы, объектные коды и исполняемые коды приложения) сохраняются вместе в одном и том же архивном файле. Выбор опции "**сжатие**" сокращает размер архива проекта.

#### ▬ **Элементы библиотеки**

Элементы библиотек ISaGRAF могут быть сохранены индивидуально. Все компоненты библиотечного элемента (техническое замечание, определение, интерфейс, исходные коды ...) сохраняются вместе в одном и том же архивном файле.

#### ▬ **Общие данные**

Команда "**Инструменты / Архив / Общие данные**" окна Управления Проектом позволяет пользователю сохранять или восстанавливать данные "общего диапазона" в ISaGRAF Workbench. Эта команда не действует на библиотеки ISaGRAF. Ниже дан список файлов, которые могут быть скопированы этой командой:

**common.eqv** ..... общие макроопределения

**oem.bat** ..... пользовательский командный файл MS-DOS

Эти файлы сохраняются один за одним на резервный диск, в их оригинальной форме. Соответствующие архивные файлы никогда не сжимаются.

### A.23.2 Опции

Путь архивов ISaGRAF появляется внизу диалога. Нажмите кнопку "Смотреть" чтобы посмотреть диск и выбрать другой диск или директорию.



Когда установлена эта опция, все архивные файлы, создаваемые во время процедуры "**Копирования**", сжимаются. Эта опция очень полезна для уменьшения размера большого архивного файла проекта, и сохранения его только на одной дискете. Сжатие архива в основном не требуется для компонентов библиотеки. Менеджер архива ISaGRAF автоматически распознает статус архивного файла (со сжатием или нет), во время восстановления архива. Это подразумевает, что опция "**сжатие**" не имеет эффекта для процедуры "**Восстановить**".



### A.23.3 Дублирование и восстановление

Список "**Система**" показывает объекты, существующие на ISaGRAF Workbench, установленные на жестком диске. Список "**Архив**" показывает объекты, сохраненные на указанном архивном диске и каталоге.

#### ▬ **Дублирование**

Сохранение объекта в архиве достигается выбором объекта в списке **слева** (объекты ISaGRAF Workbench) и нажатием кнопки "**Сохранить**". Более одного объекта может быть выбрано в списке. Кнопка "**Сохранить**" запрещается, когда элемент выбран в списке **справа** (режим восстановления).

#### ▬ **Восстановление**

Копирование объекта из архива в ISaGRAF Workbench достигается путем выбора объекта в списке **справа** (архивные объекты), и нажатием кнопки "**Восстановить**". Более одного объекта может быть выбрано в списке. Кнопка "**Восстановить**" запрещается, когда элемент выбран в списке **слева** (режим дублирования).

### A.23.4 Файлы архива

Менеджер архивов ISaGRAF создает уникальный архивный файл для каждого из сохраненных объектов. Архивный файл имеет то же имя, что и объект. Его расширение определяет тип. Ниже даны используемые расширения:

.pia..... проект

**.bia**.....плата В/В  
**.iia**.....функция на языке IEC  
**.aia**.....функциональный блок на языке IEC  
**.uia**.....С функция  
**.fia**.....С функциональный блок  
**.cia**.....С функция преобразования  
**.ria**.....конфигурация В/В  
**.xia**.....оборудование В/В

## A.24 Печать готового документа

Генератор Документов ISaGRAF позволяет пользователю строить и печатать законченный документ для выбранного проекта. Окно генератора документа отображается после активизации команды **“Печать”** из меню **“Проект”** менеджера Проектов. В отличие от команд **“Печать”** из других окон ISaGRAF Workbench, Генератор Документов может быть использован для печати более чем одной компоненты проекта в одном и том же документе, с глобальным форматом и нумерацией страниц.

Команды меню **“Редактировать”** используются для определения элементов проекта, которые должны быть включены в документ. Таким путем пользователь строит **“таблицу содержания”** необходимого документа. Любая информация о проекте (программы, переменные, опции, соединения В/В ...) могут быть вставлены в проектный документ. Никакой информации из другого проекта или из библиотек ISaGRAF не может появиться в этом документе.



Команда **“Файл / Печать”** генерирует документ и посылает его на принтер, следуя определенной таблице содержания. Работа **“Печать”** может занять несколько минут для построения и форматирования документа. Рекомендуется подождать, пока не завершится **“Работа Печати”** в окне Генератора Документов ISaGRAF, перед запуском других команд ISaGRAF Workbench. Построение документа целиком может потребовать много места на диске. Сообщение об ошибке будет выведено, если весь диск заполнен. В таком случае пользователь должен будет либо освободить дисковое пространство, удалив файлы, либо уменьшить объем работы печати. Появляется окно диалога, когда запускается команда **“Печать”**. Это позволяет пользователю вводить замечание, описывающее фактическую команду печати. Эти замечания сохраняются в файле предьстории, и будут напечатаны на первой странице любого будущего документа (включая настоящий).

### A.24.1 Настройка таблицы содержания

Меню **“Редактировать”** содержит команды для определения **“Таблицы Содержания”** документа. Выбор команд позволяет пользователю использовать таблицу, заданную по умолчанию (со всеми компонентами проекта), построить специальную таблицу (только с определенными компонентами) или перенести пункты в таблицу и модифицировать их.



#### **Список по умолчанию**

Команда **“Список по умолчанию”** из меню **“Редактировать”** определяет стандартную таблицу содержания для документа, которая включает все компоненты проекта. Стандартная таблица состоит из:

- Дескриптор проекта
- Иерархическое дерево (связи между программами)
- Исходные коды для любой программы
- Файл регистрации для любой программы
- Общие определения
- Глобальные определения



- Локальные определения для любой программы
- Глобальные переменные
- Локальные переменные для любой программы
- Опции приложения
- Соединения В/В
- Списки переменных
- Таблицы преобразования
- Сжатые перекрестные ссылки
- Детальные перекрестные ссылки
- Сводка описания
- Карта адресов сети
- Предыстория изменений

Содержимое таблицы может быть сохранено на диске с помощью команды "**Файл / Сохранить**". Эта команда окрашена серым, когда генератор документов запускается из редактора ISaGRAF для печати одного документа.



### **Вырезать и вклеить**

Используйте команды "**Редактор/ Вырезать**" и "**Редактор/ Вставить**" для передвижения пунктов в списке при настройке таблицы. Генератор Документов позволяет множественное выделение, так что группа пунктов может быть вырезана и вклеена.



### **Очищение таблицы**

Команда "**Редактор/ Очистить**" очищает таблицу содержания, так что она может быть в целом перестроена, путем вставки одного пункта.



### **Вставка пунктов в таблице**

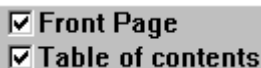
Когда запущена команда "**Редактор/ Вставить**", появляется следующее окно диалога. Оно позволяет пользователю вставлять пункты (компоненты проекта) в таблицу содержания.

Для пунктов, относящихся к программе, используйте меню "**Программа**" для выбора имени программы. Нажмите кнопку "**Добавить**" для вставки выбранного пункта в таблицу содержания. Один и тот же пункт может появиться в таблице только один раз.

## **A.24.2 Опции**

Команды меню "**Опции**" используются для определения и настройки формата генерируемого документа.

Другие опции доступны через кнопки окна Генератора Документов:



Когда установлена опция "**Передняя страница**", головная страница печатается в начале документа, содержащая название проекта и историю распечаток. Если эта

опция не установлена, первый символ, который должен быть напечатан начинается на первой странице.

Когда установлена опция "Содержание", в конце документа печатается содержание.

Изначально обе опции не установлены.



## SFC схемы

Опция "Уровни SFC" указывает системе печатать для каждой SFC программы сначала уровень 1 (схемы и комментарии), а затем уровень 2 SFC программирования. Когда эта опция не выбрана, уровни 1 и 2 появляются вместе на одной распечатке.



## Формат страницы

Команда "Формат страницы" меню "Опции" используется для определения основных параметров, которыми оперирует Генератор Документа во время. Следующие параметры могут быть указаны

- **Левое поле:** (1 или 2 сантиметра, или нет поля)
- **Граница страницы:** Если выбрана эта опция, то вокруг любой напечатанной страницы рисуется граница.



## Шаблон титула страницы

Команда "Заголовок страницы" меню "Опции" используется для определения содержимого прямоугольного титула, напечатанного внизу каждой страницы. Стандартное расположение этого прямоугольника показано ниже:

	Text1	IsaGRAF - Project 'PrName'	date
	Text2		
	Text3	User defined title	page

Первая строка основного титула (с именем проекта IsaGRAF), текущая дата и номер страницы автоматически генерируются Менеджером Документа, и не могут быть изменены.

Три строки текста с левой стороны прямоугольника (текст1, текст2, текст3) и вторая строка основного титула определяются пользователем. Пользователь также может изменить лого, напечатанное в прямоугольнике слева. Для использования другого лого пользователь должен определить путь к файлу bitmap имиджа (.BMP). Имидж может иметь любой размер. Он будет вытянут или сжат в соответствии с точными размерами печатаемой страницы. Новый определенный имидж можно увидеть щелкнув мышкой на лого в окне диалога. Файл имиджа должен быть на диске (в указанном каталоге и с указанным именем), когда запускается команда "Печать".



## Выбор символьных шрифтов

Команды "Шрифт текста" и "Шрифт заголовка" меню "Опции" используется для определения шрифтов символов, используемых во время печати текста и титулов для любого пункта документа. Размер и стиль символов также может

быть выбран для текста и титулов. Выбор шрифта делается в стандартном окне диалога, определенным Windows. Любой текст (текстовые программы, имена внутри диаграмм ...) будут напечатаны выбранным размером, стилем и шрифтом символов. Только титулы будут напечатаны шрифтом, выбранным для титулов. Вот пример настроенной распечатки:

Если шрифты символов не определены, то будет использован стандартный шрифт принтера для любого текста, со следующими стилями:

- Стиль "normal" для текстов и имен внутри диаграмм
- Стиль "bold" для титулов

## A.25 Парольная защита

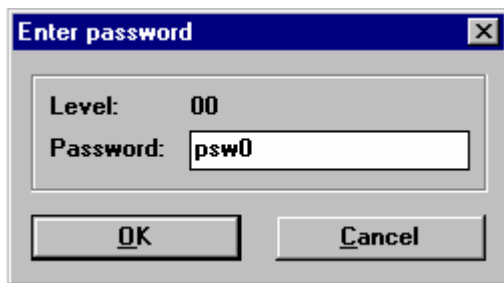
ISaGRAF Workbench включает систему полной защиты данных, которая позволяет пользователю защищать паролем проекты и элементы библиотеки. Библиотечный элемент может быть конфигурацией В/В, платой В/В или сложным оборудованием, функцией или функциональным блоком, написанным на языке IEC, “С” функцией, функциональным блоком или функцией преобразования. Защищенная паролем база данных предназначена для одного проекта или библиотечного элемента, и не может быть разделена между несколькими.

### ▬ Уровни защиты

Внутри одного проекта или библиотечного элемента пользователь может определить до **16** уровней доступа, с соответствующими различными паролями. Уровни доступа отсортированы в иерархической системе. Они пронумерованы от **0** до **15**. Наивысший уровень доступа пронумерован как **0**. Когда пользователь знает пароль, он имеет доступ ко всем пунктам, защищенным на соответствующем уровне доступа, плюс все пункты, защищенные более низкими уровнями. Каждая элементарная команда, или данные проекта, или библиотечный элемент может быть отдельно защищен уровнем доступа. Например, команда “Создать код приложения” из меню ISaGRAF может быть защищена отдельно. Элементарные данные могут быть программой, списком опций, техническим замечанием библиотечного элемента и т.д...

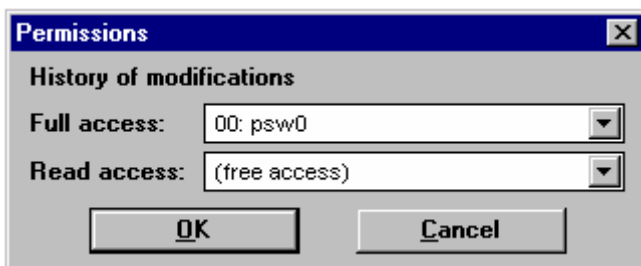
### ▬ Определение парольной защиты

Команда “Установить пароль” из меню ISaGRAF используется для определения паролей и уровней доступа для одного проекта или одного библиотечного элемента. Эта команда вызывается из меню ISaGRAF Project Manager (для проекта), или из Менеджера Библиотек ISaGRAF (для библиотечного элемента). Пароль не требуется, когда эта команда запускается в первый раз. Если пароли уже определены, то пользователь должен ввести пароль самого высокого уровня, который он знает, перед выполнением этой команды. Более верхний уровень паролей и защищенных пунктов тогда не могут быть изменены. Команда “Установить пароль” позволяет пользователю определить пароли соответствующие различным уровням доступа, и защищать элементарные команды или данные с определенными уровнями. Пароли (соответствующие уровням защиты) вводятся двойным нажатием на линию верхнего списка. Следующий диалог используется для ввода пароля:



Список в нижней части показывают различные пункты (данные или функции), которые могут быть защищены, и текущий уровень защиты прикрепленный к правам "доступ чтения" или "полный доступ". Присвоение уровню защиты права "чтения" позволяет вам запретить пользователю без достаточных прав даже открывать или печатать документ.

Нажмите дважды на линию в нижнем списке для установки прав выбранного пункта или данных. Открывается следующий диалог:



Оба права могут быть установлены для свободного доступа или для уровня защиты определенного паролем. Полное право доступа не может быть получено для уровня с меньшим приоритетом, чем выбранный для чтения доступ.

Заметим, что для некоторых документов, видных при использовании ISaGRAF Workbench, таких как дескриптор проекта, доступ чтения не может быть защищен паролем.

## ⇒ **Доступ к защищенным данным**

Когда запускается Workbench, пароль или имя пользователя не запрашиваются. Каждый раз, когда пользователь хочет иметь доступ к защищенным данным или функциям, он должен ввести требуемый пароль в окне диалога.

Если пользователь вводит требуемый пароль (или пароль назначенный для самого высокого уровня защиты), то он может продолжать нормально. Каждый раз, когда пароль вводится пользователем, он сохраняется в памяти, так что позже пользователь не должен будет вводить его снова. Сохраненные пароли удерживаются каждый раз, когда средство ISaGRAF вызывается из другого средства ISaGRAF (например, Project Manager запускает Program Manager). Сохраненные пароли теряются, когда последнее оставшееся окно ISaGRAF закрывается. Пароли, введенные во время редактирования проекта, или во время

использования Library Manager, или используя Менеджер Архива, не могут быть разделяемыми. Если пользователь вводит плохой пароль, то он не может запустить выбранную функцию.

### ⇒ **Связи с менеджером архива**

Когда объект сохраняется (проект или библиотечный элемент) на архивный диск, то вызывается пункт защиты данных, названный "**Сохранить в Архив**". Это соответствует системе защиты данных, подключенной к объекту на Workbench (жесткий диск). Тесты не проводятся на системе защиты данных объекта на архивном диске, если он уже существует. Команда "**Сохранить**" Менеджера Архивов ISaGRAF сохраняет информацию защиты данных с объектом на архивный диск.

Когда восстанавливается объект, который уже существует на Workbench (жесткий диск), то вызывается пункт защиты данных, названный "**Переписать Архив**". Это соответствует системе защиты данных, подключенной к объекту на Workbench (жесткий диск). Тесты не проводятся на системе защиты данных объекта на архивном диске. Если эта команда достоверна, то тогда восстановленная информация защиты данных заменит существующую на жестком диске.

### ⇒ **Установка индивидуальных защит для каналов В/В**

Система разработки ISaGRAF обеспечивает полную систему защиту данных основанную на иерархических паролях. Соединения В/В могут быть глобально защищены паролем. Дополнительно, ISaGRAF позволяет вам установить индивидуальную защиту на любой канал В/В. Это подразумевает, что:

- пароли всегда определяются в системе определения пароля (используйте команду "**Проект / Установить пароль**" окна Управление Проектами) так, что имеются индивидуальные уровни защиты.

- вы используете для индивидуальной защиты уровни защиты с более высоким приоритетом по сравнению с глобальной защитой В/В.

Когда канал В/В имеет индивидуальную защиту, маленькая иконка нарисована рядом с его именем в окне соединений В/В:

Используйте команды "**Установить защиту**" и "**Уничтожить защиту**" меню "**Редактор**", чтобы установить или удалить индивидуальную защиту для выбранного канала. Обе команды предлагают вам ввести действительный пароль так, что уровень защиты может быть присоединен к каналу. Затем, каждый раз, когда вы хотите изменить соединение с каналом вы должны ввести пароль достаточного уровня приоритета.

Предупреждение: Если канал защищен паролем, то соответствующий пароль удаляется из системы защиты, и если не определено пароля с более высоким уровнем, соединение с каналом не может больше быть изменено, без нового пароля с достаточным уровнем.

## A.26 Улучшенная техника программирования

Эта глава содержит больше информации об ISaGRAF Workbench и целевой системе. Рекомендуется, чтобы пользователь был знаком со средствами и методами ISaGRAF перед чтением этого раздела.

### A.26.1 Немного больше о средствах ISaGRAF

Во время использования средств редактирования ISaGRAF, пользователь может нажать **правую кнопку мышки** для открытия всплывающего меню, которое содержит основные команды редактирования. Меню открывается в текущей позиции курсора. Это очень полезно для уменьшения мышинных операций во время выполнения команд вырезки и вставки.

Средства ISaGRAF поддерживают **многократное выполнение**. Хотя некоторые средства не могут быть открыты дважды для редактирования того же самого документа, возможно открытие различных окон с тем же средством и редактирование различных объектов как параллельные операции.

Для нахождения информации о графических кнопках в меню средств доступны другие команды. Двойной щелчок мышкой на свободном месте меню средств для отображения содержимого меню средств в виде всплывающего меню. Курсор мышки, оставленный на графической кнопке, отображает соответствующую текстовую команду.

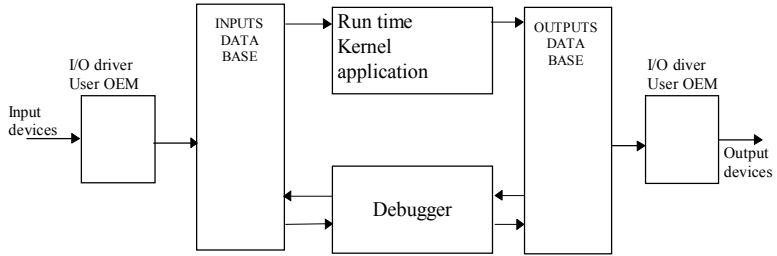
### A.26.2 Заблокированный В/В и виртуальный В/В

Определение платы В/В как **виртуальной**, отключает обработку физических каналов В/В. Когда плата определена как виртуальная, операции ядра ISaGRAF не изменяются. Все различие в том, что входные датчики не считываются и выходные устройства не обновляются. В этом режиме возможно использовать отладчик ISaGRAF для изменения входных значений. Атрибут **Виртуальный** применяется ко всей плате целиком. Это программируется во время определения платы В/В, перед генерацией кода приложения. Атрибут **виртуальный** - это **статическое** свойство и сохраняется, когда приложение останавливается и перезапускается.

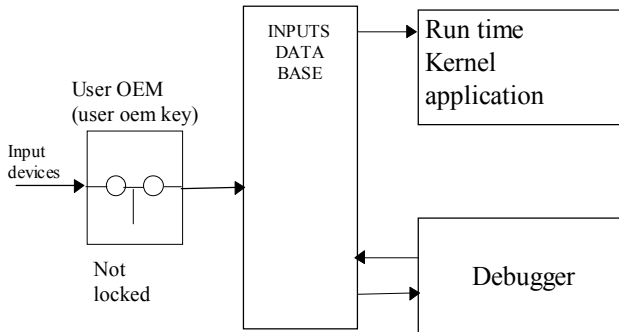
Общие положения главных свойств управления В/В

	<i>Виртуальный атрибут</i>	<i>Команда блокировки</i>
средство выбора	соединение платы В/В	отладчик
определение	статичный	динамичный
режим выбора	плата	переменная
приложение	проверка достоверности и тесты	обслуживание

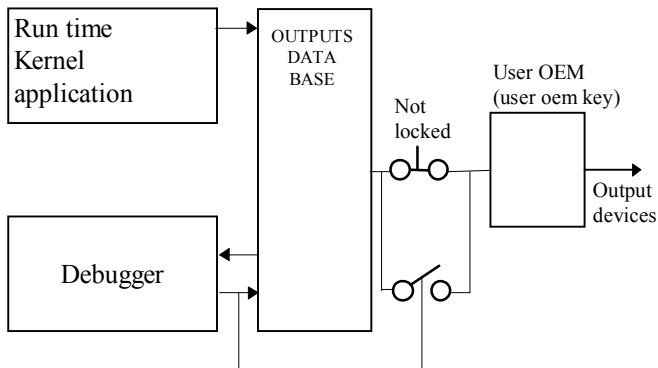
Следующая схема объясняет поток данных В/В между задачами ISaGRAF:



Когда входная переменная заблокирована, различные доступы к базе данных не изменяются, но входное устройство отключено. Входные значения могут быть установлены отладчиком и обработаны ядром ISaGRAF:

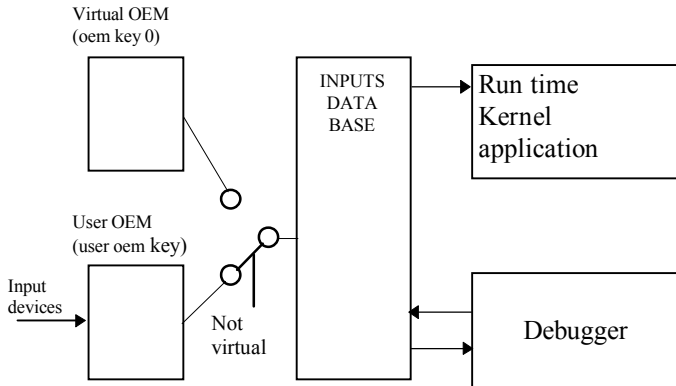


Когда выходная переменная заблокирована, выполняемое ядро и выходной драйвер не соединены. В этом случае, доступ к выходному устройству еще возможен через выходной драйвер, при помощи отладчика ISaGRAF:

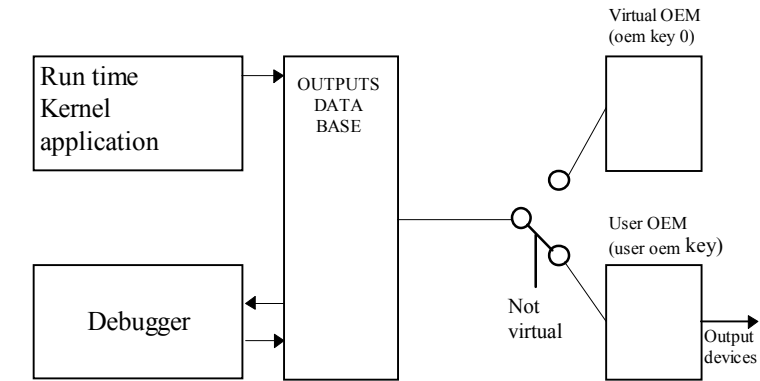




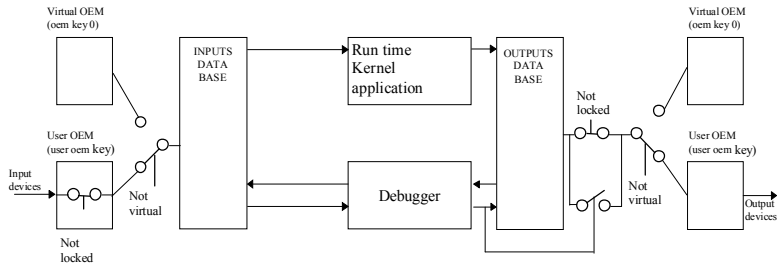
Когда устанавливается виртуальный атрибут для входа, входная база данных и соответствующие входные устройства не соединены. Виртуальный драйвер В/В заменяет реальный драйвер.



Установка виртуального атрибута следует тем же самым правилам, что для входных плат, что для выходных. Для выходных плат ядро ISaGRAF обновляет выходную базу данных. Эта база данных и соответствующие выходные устройства, тем не менее, не соединены. Виртуальный драйвер В/В заменяет реальный.



Обобщим все возможности:



### A.26.3 Проверка достоверности связи PC-PLC

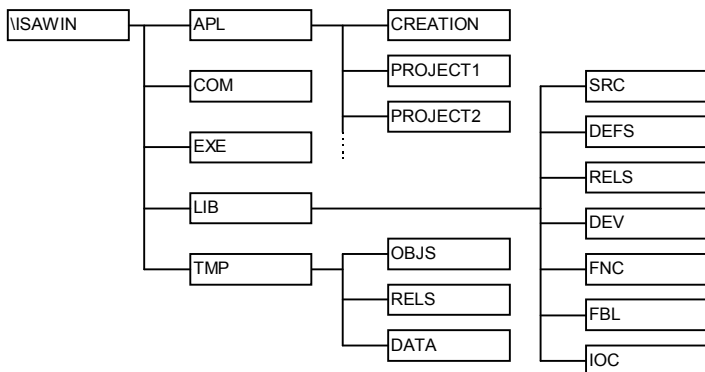
Большинство проблем, касающихся плохой связи между ISaGRAF Workbench и целевым PLC, представлены в окне отладчика статусным сообщением **“нет связи”**. Перед выполнением любых диагностических тестов должна быть проведена проверка достоверности связи, когда нет активных приложений на целевом PLC. Таким методом последовательная связь может быть проверена сама по себе, изолированно от эффектов выполнения приложения.

Язык “С”, используемый для описания функций преобразования и С функций, разрешает прямой доступ к целевой системе. Ошибки программирования в таком программном компоненте могут вызывать системные ошибки или некорректное поведение системы ISaGRAF. Такие проблемы могут случаться, когда драйверы В/В разрабатываются с Опцией ISaGRAF **IO development tool**. Системные ошибки, например, могут произойти, если плата В/В подключена на неправильный адрес шины. Следующая таблица дает синтетическое обобщение диагностики ошибок:

статус	контекст	диагностика
“нет связи” (перед загрузкой)		<ul style="list-style-type: none"> <li>- цель не запущена</li> <li>- нет кабеля / плохой кабель</li> <li>- неправильные параметры связи</li> <li>- цель ISaGRAF плохо установлена</li> </ul>
“нет связи” (после загрузки)	цикл за циклом режим запуска	<ul style="list-style-type: none"> <li>- неправильная конфигурация В/В</li> <li>- сбой системы</li> </ul>
	реальное время режим запуска	<ul style="list-style-type: none"> <li>- неправильная конфигурация В/В</li> <li>- сбой системы (из-за С программирования)</li> </ul>
“нет приложения”		<ul style="list-style-type: none"> <li>- приложение не загружено</li> <li>- приложение не запущено (из-за С программирования)</li> <li>- несовпадение Intel/Motorola</li> <li>- неправильная целевая версия</li> </ul>

## A.26.4 Каталоги ISaGRAF

ISaGRAF Workbench работает со специализированной структурой каталога диска. Корневой каталог в этой архитектуре определяется пользователем во время установки ISaGRAF. Имя по умолчанию для корневого каталога - **ISAWIN**. Это стандартная архитектура диска, созданная программой установки:



Это стандартные подкаталоги ISaGRAF

КАТАЛОГ	СОДЕРЖИМОЕ
APL	корневой каталог для проектов ISaGRAF каждый проект соответствует одному подкаталогу, который содержит все данные проекта
COM	данные из разряда "общие" Данные могут быть использованы любым проектом
EXE	программы ISaGRAF и файлы подсказок
LIB	библиотеки ISaGRAF: - списки элементов - параметры или интерфейс для каждого элемента - технические замечания
LIB\IOC	исходные коды для конфигураций В/В
LIB\FNC	исходные коды функций, написанных на языке IEC
LIB\FBL	исходные коды функциональных блоков, написанных на языке IEC
LIB\SRC	исходные коды для преобразований и С функций
LIB\DEFS	исходные заголовки для преобразований и С функций
LIB\RELS	объектные коды преобразований и С функций
LIB\DEV	командные файлы для разработки "С" библиотек makefiles, списков связей и т.д.
TMP	временные файлы: TMP подкаталоги зарезервированы для Генератора Кодов ISaGRAF и не может быть уничтожен.

Подкаталоги могут быть перенесены на другое место на диске. Когда у пользователя нестандартная архитектура, пути к подкаталогам должны быть

определены в разделе **WS001** в файле инициализации **ISA.ini**, в подкаталоге **EXE**. Здесь даны вхождения в разделе **WS001**:

<b>Isa</b>	корневой каталог для архитектуры ISaGRAF
<b>IsaExe</b>	корневой каталог для программ и файлов подсказок ISaGRAF
<b>IsaApl</b>	корневой каталог для проектов ISaGRAF
<b>IsaTmp</b>	каталог для временных файлов
<b>IsaSrc</b>	каталог для библиотечных исходных кодов
<b>IsaDefs</b>	каталог для библиотечных исходных заголовков

Заметьте, что если вы меняете IsaTmp вхождение на другой каталог, то вы должны создать подкаталоги OBJS, RELS и DATA в новом каталоге.

Следующий пример использует вхождения раздела **WS001** для переопределения стандартной дисковой архитектуры ISaGRAF:

```
;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
```

Когда вы хотите добавить функцию “C” или функциональный блок в целевое приложение ISaGRAF, то каталог **ISAWIN\LIB\DEV** используется для сохранения файлов разработки: командных файлов, мейкфайлов, карт и т.д. **ISAWIN\LIB\RELS** используется для сохранения объектных файлов, генерируемых во время “C” компиляции, и для ISaGRAF “C” библиотек, требуемых для операции LINK.

## A.26.5 Символы приложения

К каждому объекту приложения ISaGRAF ссылаются по имени (введенному во время объявления переменной ) и по внутреннему **виртуальному адресу**, вычисленному генератором кодов. Виртуальный адрес переменной - это не ее **сетевой адрес**, введенный во время объявления переменной. Виртуальные адреса используются для связанной работы, и специальных “C” приложений, использующих опцию **IO development tool**. Когда генератор кодов ISaGRAF запущен, он создает ASCII файл с логическим соответствием между именами и виртуальными адресами для всех объектов (переменные, программы, шаги...) проекта. Этот файл может быть легко запрошен из любого приложения пользователя для получения информации о статической базе данных ISaGRAF. Файл называется “**APPLI.TST**” и располагается в каталоге проекта ISaGRAF: “**ISAWIN\APL\prname**” (prname - имя проекта). Этот раздел описывает детальный формат файла “**APPLI.TST**”. Основные нотации, используемые для последующего описания, показаны ниже:

<b>VA</b>	виртуальный адрес
<b>ATTR</b>	атрибут переменной

**USR** "C" функция

Возможные значения для атрибутов переменной показаны ниже. Такие значения содержатся в полях "**атрибутов**" ("**attributes**"):

**+X** внутренняя переменная  
**+C** внутренняя переменная только на чтение  
**+I** входная переменная  
**+O** выходная переменная

Все числа, исключая виртуальные адреса, выражаются как десятичные целые. Виртуальные адреса (**VA**) выражаются как шестнадцатеричные числа их 4 цифр, и предваряются символом "!". Например:

123 это десятичное значение  
!A003 это шестнадцатеричный виртуальный адрес

Основная структура файла "**APPLI.TST**" показана ниже. Файл структурирован как список блоков. Блок - это список записей. Каждая запись описывается на одной строке текста. Каждый блок начинается с заголовка, помещенного на одной строке текста.

начало блока  
описание блока  
конец блока

Основной синтаксис одного блока показан ниже:

```
@ <block_name> <arguments>  
#record...  
#record...  
...
```

Структура первого блока, содержащего главную информацию о приложении, показана ниже:

```
@ISA_SYMBOLS,<appli_crc>  
#NAME,<appli_name>,<version>  
#DATE,<creation_date>  
#SIZE,G=<nbprg>,S=<nbstep>,T=<nbtra>,L=0,P=<nbpro>,V=<nbvar>  
#COMMENT,cj international
```

**appli\_crc**..... контрольная сумма символов приложения  
**appli\_name**..... имя приложения  
**version**..... номер версии ISaGRAF  
**creation\_date**..... дата генерации приложения  
**nbprg** ..... число программ  
**nbstep**..... число шагов SFC  
**nbtra**..... число транзакций SFC  
**nbpro** ..... число используемых функций "C"  
**nbvar**..... общее число переменных

Структура последнего блока, который указывает конец файла, показана ниже:

```
@END_SYMBOLS
```

Структура блока, используемого для описания программ приложения, показана ниже:

```
@PROGRAMS, <nbprg>  
#<va>, <name>  
#...
```

**nbprg** .....число программ, определенных в этом блоке

**va**..... виртуальный адрес программы

**name** ..... имя программы

Структура блока, используемого для описания шагов SFC приложения, показана ниже. Заметьте, здесь один виртуальный шаг, определенный для каждой не SFC программы:

```
@STEPS, <nbsteps>  
#<va>, <name>, <father>  
#...
```

**nbsteps** .....число шагов, определенных в этом блоке

**va**..... виртуальный адрес шага

**name** ..... имя шага

**father**..... виртуальный адрес родителя

Структура блока, используемого для описания SFC транзакций приложения, показана ниже:

```
@TRANSITIONS, <nbtrans>  
#<va>, <name>, <father>  
#...
```

**nbtrans** .....число транзакций, определенных в этом блоке

**va**..... виртуальный адрес транзакции

**name** ..... имя транзакции

**father**..... виртуальный адрес родителя

Структура блока, используемого для описания логических переменных приложения, показана ниже:

```
@BOOLEANS, <nb_boo>  
#<va>, <name>, <attr>, <program>, <eq_false>, <eq_true>  
#...
```

и если число переменных превосходит 4095:

```
X# (1.<varno>), <name>, <attr>, <program>, <eq_false>, <eq_true>
```

**nb\_boo**.....число переменных в этом блоке  
**va**.....виртуальный адрес переменной  
**varno**.....диапазон адреса  
**name** .....имя переменной  
**attr**.....атрибут переменной  
**program**.....виртуальный адрес программы родителя  
.....или "I0000" для глобальной переменной  
**eq\_false** .....строка, используемая для значения ложь (false)  
**eq\_true**.....строка, используемая для значения истина (true)

Структура блока, используемого для описания аналоговых переменных приложения, показана ниже:

```
@ANALOGS, <nb_ana>
#<va>, <name>, <attr>, <program>, <format>, <unit>
#...
```

и если число переменных превосходит 4095:

```
X# (2.<varno>), <name>, <attr>, <program>, <eq_false>, <eq_true>
```

**nb\_ana** .....число переменных в этом блоке  
**va**.....виртуальный адрес переменной  
**varno**.....диапазон адреса  
**name** .....имя переменной  
**attr**.....атрибут переменной  
**program**.....виртуальный адрес программы родителя  
.....или "I0000" для глобальной переменной  
**format** ..... = "I" для целой переменной  
..... = "F" для вещественной переменной  
**unit**.....строка блока

Структура блока, используемого для описания переменных таймера приложения, показана ниже:

```
@TIMERS, <nb_tmr>
#<va>, <name>, <attr>, <program>
#...
```

и если число переменных превосходит 4095:

```
X# (3.<varno>), <name>, <attr>, <program>, <eq_false>, <eq_true>
```

**nb\_tmr** ..... количество переменных в этом блоке  
**va**.....виртуальный адрес переменной  
**varno**.....диапазон адреса  
**name** .....имя переменной  
**attr**.....атрибут переменной (всегда +X: внутренняя)  
**program**.....виртуальный адрес программы родителя  
.....или "I0000" для глобальной переменной

Структура блока, используемого для описания переменных сообщения приложения, показана ниже:

```
@MESSAGES, <nb_msg>  
#<va>, <name>, <attr>, <program>, < max_len>  
#...
```

и если число переменных превосходит 4095:

```
X# (4.<varno>), <name>, <attr>, <program>, <eq_false>, <eq_true>
```

**nb\_msg**.....число переменных в этом блоке  
**va**.....виртуальный адрес переменной  
**varno**.....диапазон адреса  
**name**.....имя переменной  
**attr**.....атрибут переменной  
**program**.....виртуальный адрес программы родителя  
.....или "10000" для глобальной переменной  
**max\_len**.....максимальная длина (объявленный диапазон)

Структура блока, используемого для описания "С" функций, примененных в приложении, показана ниже:

```
@USP, <nb_usp>  
#<va>, <name>  
#...
```

**nb\_usp**.....число С функций в этом блоке  
**va**.....виртуальный адрес С функции  
**name**.....имя С функции

Структура блока, используемого для описания экземпляров функционального блока "С", примененных в приложении, показана ниже:

```
@FBINSTANCES, <nb_fb>  
#<va>, <inst_name>, <fb_name>  
#...
```

**nb\_fb**.....число экземпляров функционального блока "С" в этом блоке  
**va**.....виртуальный адрес экземпляра функционального блока "С"  
**inst\_name**.....имя экземпляра функционального блока "С"  
**fb\_name**.....имя ссылки функционального блока "С"

## A.26.6 Ограничения ISaGRAF "LARGE" (WDL) workbench

Здесь даны некоторые ограничения для объектов, используемых в ISaGRAF workbench. Конечно, имеется много других практических ограничений из-за конфигурации используемого компьютера (доступной памяти и дискового пространства), и возможностей целевой системы ISaGRAF (доступной памяти,



доступных аппаратных и программных ресурсов...). Данные ниже числа - это абсолютные ограничения, которые не могут быть превышены.

### ⇒ **Для проекта:**

<i>Объект</i>	<i>Максимум</i>	<i>Замечания</i>
Программы	255	группируя основную, подпрограммы и дочерние
Уровни иерархии	20	

Число проектов, установленных на Workbench, ограничивается только доступным пространством на жестком диске

### ⇒ **Для имен:**

<i>Имя для:</i>	<i>Максимум</i>	<i>Замечания</i>
Проект	8 символов	
Программа	8 символов	
Переменная	16 символов	+60 символов на комментарий
Определенная метка слова	16 символов	
Определенный эквивалент	255 символов	+60 символов на комментарий
Таблица преобразования	16 символов	
Список переменных	16 символов	
функция / функц.блок (lib)	8 СИМВОЛОВ	налагается на функции С, функциональные блоки С или функции написанные на языках IEC
Параметр функции (lib)	16 СИМВОЛОВ	налагается на функции С, функциональные блоки С или функции написанные на языках IEC
Плата В/В	8 символов	
Конфигурация В/В	8 символов	
оem параметр платы	16 символов	
Функция преобразования	8 символов	

### ⇒ **Редактирование (для одной программы):**

<i>Объект</i>	<i>Максимум</i>	<i>Замечания</i>
Строки SFC	600	
Колонки SFC	20	
Шаги SFC	4095	для всего проекта, группируя шаги, иницирующие шаги, начальные и завершающие шаги
Транзакции SFC	4095	для всего приложения
LD/FBD редактирование	200 кол 2000 ряд	это размер области редактирования в ячейках.
Quick LD редактирование	без ограничен	ограничения налагаются мощностью PC

IL метки 251 в той же программе IL  
 Редактирование текста 40KBytes

⇒ **Для словаря (для одного проекта):**

Объект	Максимум	Замечания
Логические переменные	65535	
Аналоговые переменные	65535	группируя целые и вещественные переменные
Таймеры	65535	
Переменные сообщения	65535	
Определенные слова	4095	в том же списке (том же блоке)
Определенные слова	255	используемые в той же программе
Таблицы преобразования	127	используемые в приложении
Точки в одной таблице	32	определенные в той же самой таблице преобразования

Ограничения даны для максимального количества булевских, аналоговых и строковых групп переменных, строковых, входных и выходных переменных. Они также включают все скрытые временные или размещенные компилятором переменные. Количество переменных редактируемых вместе (одного типа, одной области действия), в редакторе словаря не может превышать 16000. В зависимости от конфигурации PC, ограничение может быть меньше 16000. Приложение не может идти на целевой задаче ISA GRAF V1.21 или ранее если суммарное количество переменных одного типа больше 4095. Стандартная связь по "Modbus" использующая сетевые адреса не может быть использована если количество переменных одного типа превосходит 4095.

⇒ **Соединения В/В:**

Объект	Максимум	Замечание
Платы ВВ	256	определены для одного приложения (платы сложного оборудования)

Количество плат В/В включая одиночные платы и сложное оборудование не превышает 256.

Каналы ВВ	128	на одной плате
-----------	-----	----------------

⇒ **Для библиотек:**

Объект	Максимум	Замечания
Функции (язык IEC)	255	Установленных вместе в библиотеке
Функциональные блоки (язык IEC)	255	Установленных вместе в библиотеке

Функции С	255	Установленных библиотеке	вместе	в
Функциональные блоки С	255	Установленных библиотеке	вместе	в
Экземпляры функциональных блоков	4095	Для того же типа функционального блока в том же приложении		
Входные параметры функции	31	Это применимо к функциям С и к функциям, написанным на языке IЕС		
Параметры функционального блока	32	свободно распределяются между входными и выходными параметрами. По меньшей мере требуется 1 выходной параметр		
Функция преобразования	128	Установленных библиотеке	вместе	в
Конфигурации В/В	255	Установленных библиотеке	вместе	в
Платы В/В	255	Установленных библиотеке	вместе	в
Сложное оборудование В/В	255	Установленных библиотеке	вместе	в
оem параметры платы	16			

## **В. Описание языка**

## В.1 Архитектура проекта

Проект ISaGRAF разделен на несколько программных модулей, называемых **программами**. Программы проекта связаны друг с другом в древовидную структуру. Программы могут быть описаны с помощью графических или текстовых языков **SFC**, **FBD**, **LD**, **ST** или **IL**.

### В.1.1 Программы

**Программа** - это логическая программируемая единица, которая описывает операции с **переменными** процесса. Программы описывают либо **последовательные** либо **циклические** операции. Циклические программы выполняются на каждом цикле целевой системы. Исполнение последовательных программ определяется динамическими правилами языка **SFC**.

Программы связаны друг с другом в иерархическое дерево. Программы, помещенные наверху иерархии, активизируются системой. Подпрограммы (нижний уровень иерархии) активизируются их родителями. Программы могут быть описаны любым из следующих графических или текстовых языков:

**Язык последовательных функциональных схем (SFC)** для программирования высокого уровня

**Flow Chart (FC)** for high level programming

**Язык функциональных блочных диаграмм (FBD)** для сложных циклических операций

**Язык релейных диаграмм (LD)** только для булевских операций

**Язык структурированный текст (ST)** только для циклических операций

**Язык инструкций (IL)** для операций низкого уровня

Одна и та же программа не может смешивать несколько языков, за исключением LD и FBD, которые могут быть скомбинированы в одной диаграмме.

### В.1.2 Циклические и последовательные операции

Иерархия программ разделена на четыре основных **секции** или группы:

<b>Begin</b>	программы, выполняемые в начале каждого цикла целевой задачи
<b>Sequential</b>	программы, определяемые динамическими правилами SFC
<b>End</b>	программы, выполняемые в конце каждого цикла целевой задачи
<b>Функции</b>	набор подпрограмм

Программы секций **Begin** и **End** описывают циклические операции и не зависят от времени. Программы секции **Sequential** описывают последовательные операции, где временная переменная явно синхронизирует основные действия. Основные программы секции **Begin** систематически выполняются в начале каждого цикла. Основные программы секции **End** систематически выполняются в конце каждого цикла. Основные программы секции **Sequential** выполняются в соответствии с динамическими правилами **SFC**.

Программы секции **Функции** - это подпрограммы, которые могут быть вызваны любой другой программой в проекте. Программы секции **Функции** могут вызывать другие программы этой же секции.

Основные программы и дочерние программы секции **Sequential** должны быть описаны при помощи языка **SFC**. Программы циклических секций (**Begin** и **End**) не могут быть описаны при помощи языка **SFC**.

Программы секции **Begin** обычно используются для того чтобы описать предварительные операции с устройствами ввода и определить значения отфильтрованных переменных верхнего уровня. Такие переменные обычно используются программами секции **Sequential**. Программы секции **End** обычно используются для того, чтобы выполнить защитные операции с переменными прежде, чем отправить их значения в устройства вывода.

### **В.1.3 Дочерние SFC программы**

Любая программа **SFC** последовательной секции может управлять другой программой **SFC**. Такие программы низкого уровня называются **дочерними SFC программами**. **Дочерние SFC программы** - это параллельные программы, которые могут быть запущены, убиты, заморожены и перезапущены своей родительской программой. Родительская программа и дочерняя программа могут быть описаны при помощи языка **SFC**. Дочерняя **SFC** программа может иметь локальные переменные и предопределенные слова.

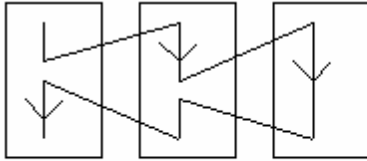
Когда родительская программа запускает дочернюю **SFC** программу, она устанавливает **SFC маркер** (активизирует) на каждом начальном шаге дочерней программы. Эта команда описывается оператором **GSTART**. Когда родительская программа убивает **SFC** дочернюю программу, она очищает все существующие маркеры на **шагах** дочерней программы. Такая команда описывается оператором **GKILL**.

Когда родительская программа замораживает **SFC** дочернюю программу, она очищает все значения в дочерней программе, и сохраняет их состояние в памяти. Такая команда описывается оператором **GFREEZE**. Когда родительская программа перезапускает **SFC** дочернюю программу, она восстанавливает все значения **SFC**, которые были очищены при замораживании программы. Такая команда описывается оператором **GRST**.

Любая программа **FC** последовательной секции может управлять другими **FC** подпрограммами. Родительская программа **FC** блокируется (ждет) пока не выполнится подпрограмма **FC**. Одновременная работа родительской программы **FC** и подпрограммы невозможна.

### **В.1.4 Функции и подпрограммы**

Исполнение подпрограмм или функций управляется их родительской программой. Исполнение родительской программы подвешивается до тех пор пока не закончит свою работу подпрограмма:



Основная программа      Подпрограммы

Любая программа любой секции может иметь одну или несколько подпрограмм. У каждой подпрограммы может быть только одна родительская программа. Подпрограмма может иметь локальные переменные и макроопределения. Для описания подпрограмм может быть использован любой язык, за исключением **SFC**. Программы секции **Функции** - это подпрограммы, которые могут быть вызваны любой другой программой в проекте. В отличие от других подпрограмм они не принадлежат никакой родительской программе. Программы секции **Функции** могут вызывать другие программы этой же секции. Функции могут быть размещены в библиотеке.

Предупреждение: Система ISaGRAF не поддерживает **рекурсивных** вызовов функций. Если программа секции **Функции** будет вызывать сама себя или будет вызываться одной из вызванных ею подпрограмм, то возникнет ошибка.

Предупреждение: Функция или подпрограмма не “запоминают” локальных значений своих локальных переменных. Функции и подпрограммы не могут вызывать функциональные блоки.

Интерфейс подпрограммы должен быть определен явно, с типами и уникальными именами каждого вызываемого и возвращаемого параметра. Для того чтобы поддержать стандарт языка **ST**, возвращаемый параметр должен иметь то же имя, что и подпрограмма.

Следующая таблица показывает, как установить значение возвращаемого параметра в теле подпрограммы, в различных языках:

**ST:** присвоить значение возвращаемому параметру, используя его имя (то же имя что и у подпрограммы):

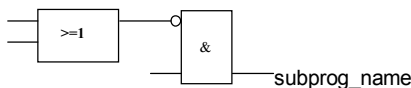
subprog\_name := <expression>;

**IL:** значение текущего результата (IL регистр)

в конце последовательности запоминается в возвращаемом параметре:

LD 10  
ADD 20 (\*значение возвращаемого параметра = 30\*)

**FBD:** установить возвращаемый параметр, используя имя:



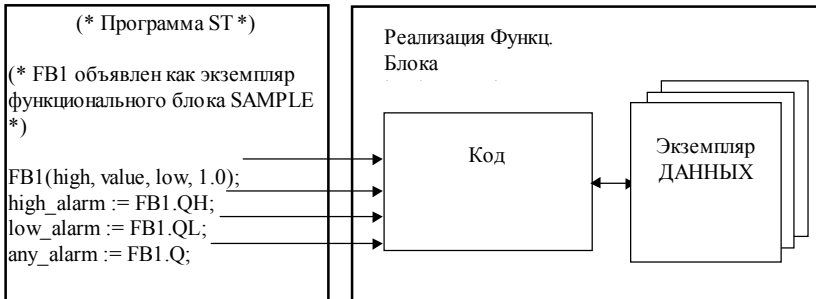
**LD:** использовать символ витка с именем возвращаемого параметра:

subprog\_name



### В.1.5 Функциональные блоки

Функциональные блоки могут использовать языки: LD, FBD, ST или IL. Локальные переменные функциональных блоков копируются для каждого экземпляра. Когда программа вызывает блок, на самом деле, вызывается экземпляр блока: вызывается тот же код, но используются данные, захваченные специально для данного экземпляра блока. Значения переменных экземпляра передаются от одного цикла к другому.



Предупреждение:

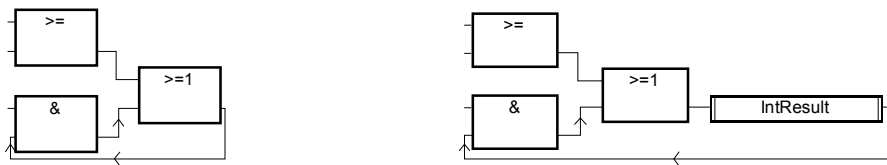
- Функциональный блок, написанный на одном из языков IEC, не может вызывать другие функциональные блоки: механизм экземпляров позволяет работать только с собственными локальными переменными блока. Вот список стандартных функциональных блоков, которые нельзя использовать внутри блоков IEC: SR, RS, R\_Trig, F\_Trig, SEMA, CTU, CTD, CTUD, TON, TOF, TP, CMP, StackInt, AVERAGE, HYSTER, LIM\_ALARM, INTEGRAL, DERIVATE, BLINK, SIG\_GEN  
SR, RS, R\_Trig, F\_Trig, SEMA, CTU, CTD, CTUD, TON, TOF, TP, CMP, StackInt, AVERAGE, HYSTER, LIM\_ALARM, INTEGRAL, DERIVATE, BLINK, SIG\_GEN

- По той же причине нельзя использовать Положительные или Отрицательные контакты или витки, или витки Установить и Сбросить.
- Функции TSTART и TSTOP для запуска и остановки таймеров не могут быть использованы внутри функционального блока для целевых задач версий 3.0x. Они работают начиная с версии 3.20.
- Если в функциональном блоке требуется виток, то прежде чем замкнуть виток, нужно использовать локальную переменную. Например:

Это не будет работать:

Это работает:





## В.1.6 Язык описания

Программа может быть описана при помощи одного из следующих графических или текстовых языков:

**Язык последовательных функциональных схем (SFC)** для программирования высокого уровня

**Язык Поточковых Диаграмм (FC)** для программирования высокого уровня

**Язык функциональных блочных диаграмм (FBD)** для сложных циклических операций

**Язык релейных диаграмм (LD)** только для булевских операций

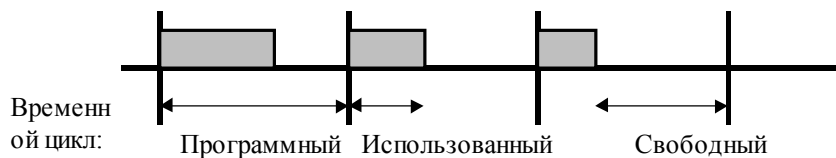
**Язык структурированный текст (ST)** для любых циклических операций

**Язык инструкций (IL)** для операций низкого уровня

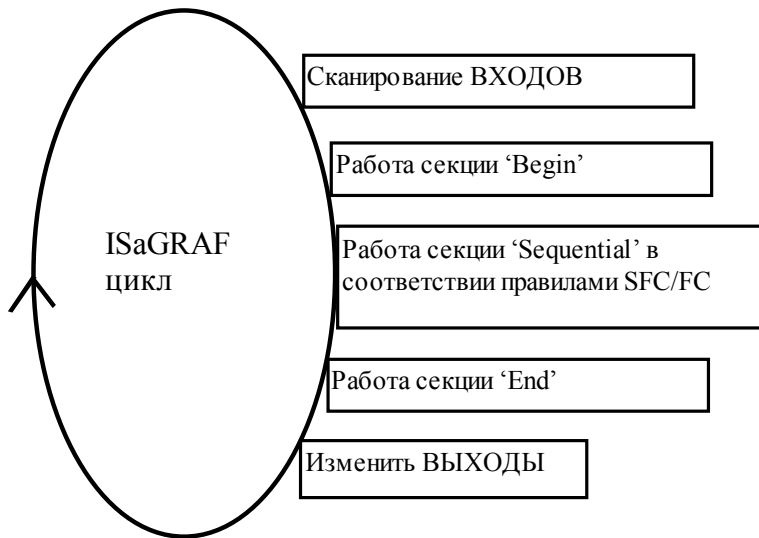
Одна и та же программа не может смешивать несколько языков. Язык программирования программы выбирается в момент создания программы и не может быть изменен в последствии. Исключением являются языки FBD и LD, которые можно комбинировать внутри одной программы.

## В.1.7 Правила исполнения

ISaGRAF - это **синхронная** система. Все операции исполняются по часам. Основная единица длительности времени называется временным циклом:



Основные операции в течение временного цикла:



Это позволяет системе:

- гарантировать, что входные переменные сохраняют свое значение в течение временного цикла,
- гарантировать, что устройства вывода изменяются не более одного раза в течение временного цикла,
- правильно работать с одними и теми же глобальными переменными из различных программ,
- оценивать и управлять временем реакции всего приложения.

## В.2 Общие объекты

Это основные особенности и общие **объекты** базы данных ISaGRAF. Такие объекты могут использоваться любым из языков **SFC, FBD, LD, ST** или **IL**.

### В.2.1 Основные типы

Любая константа, выражение или переменная, используемая в программе (написанной на любом языке) должна характеризоваться своим типом. Типы должны быть согласованы в графических операциях и текстовых выражениях. Вот основные типы программных объектов:

<b>BOOLEAN:</b>	логическая (true или false) величина
<b>ANALOG:</b>	целая или действительная (плавающая) непрерывная величина
<b>TIMER:</b>	временная величина
<b>MESSAGE:</b>	строка символов

Замечание: Переменные типа **TIMER** (временные) содержат значения не большие, чем один день, и не могут быть использованы для записи данных.

### В.2.2 Константы

Константы относятся к одному типу. Одно и то же обозначение не может быть использовано для представления констант разных типов.

#### В.2.2.1 Логические константы

Существует только две логические константы:

**TRUE** эквивалентно целому значению 1

**FALSE** эквивалентно целому значению 0

Для написания ключевых слов "true" и "false" можно использовать, как заглавные буквы, так и прописные.

#### В.2.2.2 Целые аналоговые константы

Целые константы представляются знаковыми длинными целыми (32 бита) величинами:

от **-2147483647** до **+2147483647**

Целые аналоговые константы могут быть представлены с одной из следующих **баз**. Целые константы должны начинаться с **префикса**, который определяет используемую базу:

База	Префикс	Пример
------	---------	--------

DECIMAL	(none)	-908
HEXADECIMAL	"16#"	16#1A2B3C4D
OCTAL	"8#"	8#1756402
BINARY	"2#"	2#1101_0001_0101_1101

Символ подчеркик ('\_') может быть использован для того, чтобы разделить группы цифр. Он не имеет особенного значения и используется для улучшения читаемости констант.

### V.2.2.3 Действительные аналоговые константы

Действительные аналоговые константы могут быть записаны в **десятичном** или **научном** представлении. **Десятичная точка** (':.') разделяет целую и десятичную части. Десятичную точку нужно использовать, для того чтобы отличить действительную константу от целой. Научное представление использует буквы '**E**' или '**F**' для того, чтобы отделить **мантиссу** от **экспоненты**. Экспоненциальная часть в научном представлении должна быть знаковой целой величиной от **-37** до **+37**. Ниже приведены примеры действительных аналоговых констант:

3.14159	-1.0E+12
+1.0	1.0F-15
-789.56	+1.0E-37

Выражение '**123**' не представляет действительную константу. Ее правильное представление '**123.0**'.

### V.2.2.4 Временные константы

Временные константы представляют временные значения от **0** секунд до **23h59m59s999ms**. Наименьшая допустимая единица - это миллисекунда. Стандартные временные единицы, использующиеся в константах это:

<b>Час</b>	буква <b>h</b> должна следовать за количеством часов
<b>Минута</b>	буква <b>m</b> должна следовать за количеством минут
<b>Секунда</b>	буква <b>s</b> должна следовать за количеством секунд
<b>Миллисекунда</b>	буквы <b>ms</b> должны следовать за количеством миллисекунд

Временные константы должны начинаться с '**T#**' или '**TIME#**'. Для написания ключевых слов можно использовать как заглавные буквы, так и прописные. Некоторые единицы могут быть опущены. Вот примеры временных констант:

<b>T#1H450MS</b>	1 час, 450 миллисекунд
<b>time#1H3M</b>	1 час, 3 минуты

Выражение '0' представляет аналоговую константу, а не временное значение.

### V.2.2.5 Константа - строка сообщения

Константа-сообщение представляет собой строку символов. Строка должна быть заключена в кавычки. Например:

**'THIS IS A MESSAGE'**

**'THIS IS A MESSAGE'**

Предупреждение: символ кавычка "" не может быть использован внутри строки-константы. Строка-константа должна быть записана в одной строке исходного текста программы. Ее длина не должна превышать 255 символов, включая пробелы. Пустая строка-константа представляется двумя кавычками без пробела или табуляции внутри:

" (\*это пустая строка\*)

Для представления непечатаемых символов может быть использован знак \$:

Последовательность	Значение	ASCII (hexa)	Пример
\$\$	знак '\$'	16#24	'I paid \$\$ for this'
'	кавычка	16#27	'Enter '\$Y\$' for YES'
\$L	строка вверх	16#0a	'next \$L line'
\$R	возврат каретки	16#0d	' llo \$R He'
\$N	новая строка	16#0d0a	'This is a line\$N'
\$P	новая страница	16#0c	'lastline \$P first line'
\$T	Табуляция	16#09	'name\$Tsize\$Tdate'
\$hh (*)	любой символ	16#hh	'ABCD = \$41\$42\$43\$44'

(\*)"hh" - шестнадцатеричное значение ASCII кода символа.

## В.2.3 Переменные

Переменные программы могут быть **локальными** или **глобальными**. Локальные переменные могут использоваться только одной программой. Глобальные переменные могут использоваться любой программой проекта. Имена переменных должны удовлетворять следующим правилам:

имя не может быть длиннее **16** символов

первым символом должна быть **буква**

последующими символами могут быть **буквы, цифры** или символ подчеркивания

### В.2.3.1 Зарезервированные ключевые слова

Ниже представлен список зарезервированных ключевых слов. Такие идентификаторы не могут быть использованы в качестве имен программ, переменных, "C" функций или функциональных блоков.

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, ELSIF, EN, END_CASE, END_FOR, END_FUNCTION, END_IF, END_PROGRAM, END_REPEAT, END_RESSOURCE, END_STRUCT, END_TYPE, END_VAR, END_WHILE, ENO, EQ, EXIT, EXP, EXPT,
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

Все ключевые слова, начинающиеся с подчеркика, являются внутренними ключевыми словами и не могут быть использованы в текстовых инструкциях.

### B.2.3.2 Переменные прямого представления

ISaGRAF позволяет использовать **переменные прямого представления** для того, чтобы представлять в исходном тексте программы свободные каналы. Свободные каналы - это каналы, не связанные с декларированными переменными ввода/вывода. Идентификатор переменной прямого представления всегда начинается с символа "%".

Ниже приводятся соглашения об именах переменных прямого представления для каналов одиночной платы. "s" - это номер слота на палате. "c" - это номер канала.

- %IXs.c** свободный канал платы булевого ввода
- %IDs.c** свободный канал платы целого ввода
- %ISs.c** свободный канал платы строкового ввода
- %QXs.c** свободный канал платы булевого вывода
- %QDs.c** свободный канал платы целого вывода

**%QSS.c** свободный канал платы строкового вывода

Ниже приводятся соглашения об именах переменных прямого представления для каналов комплексного оборудования. "s" - это номер слота оборудования. "b" - это индекс одной платы внутри комплексного оборудования. "c" - это номер канала.

<b>%Xs.b.c</b>	свободный канал платы булевого ввода
<b>%IDs.b.c</b>	свободный канал платы целого ввода
<b>%Is.b..c</b>	свободный канал платы строкового вывода
<b>%QXs.b.c</b>	свободный канал платы булевого вывода
<b>%QDs.b.c</b>	свободный канал платы целого вывода
<b>%QSS.b.c</b>	свободный канал платы строкового вывода

Примеры:

**%QX1.6** 6 канал 1 платы (логический вывод)

**%ID2.1.7** 7 канал 1 платы во 2 оборудовании (целый ввод)

Переменные прямого представления не могут иметь тип **real**.

### **V.2.3.3 Булевские переменные**

Булевские означает **логические**. Такие переменные могут принимать одно из двух булевых значений: **TRUE (ИСТИНА)** или **FALSE (ЛОЖЬ)**. Обычно, булевские переменные используются в логических выражениях. Булевские переменные могут иметь один из следующих **атрибутов**:

**Внутренняя:** переменная, хранящаяся в памяти, изменяемая программой

**Константа:** неизменяемая переменная, хранящаяся в памяти, с начальным значением

**Вход:** переменная, связанная с устройством ввода (обновляется системой)

**Выход:** переменная, связанная с устройством вывода

Предупреждение: При объявлении логической переменной, могут быть определены строки, заменяющие 'true' или 'false' во время отладки. Эти строки не могут быть использованы в программах, если они не введены как '**предопределенные слова**' для языка.

### **V.2.3.4 Аналоговые переменные**

Аналоговые означает - **непрерывные**. Такие переменные могут принимать значения знаковых целых или действительных (плавающих). Возможны следующие форматы:

**Integer** 32 битовое знаковое целое: от **-2147483647** до **+2147483647**

**Real** стандартное 32 битовое IEEE плавающее значение (одиночная точность)

1 знаковый + 23 бита мантииссы + 8 бит экспоненты

Значение экспоненты аналоговой переменной типа Real не может быть меньше -37 и больше +37.

Аналоговые переменные могут иметь один из следующих атрибутов:

**Внутренняя:** переменная, хранящаяся в памяти, изменяемая программой

**Константа:** неизменяемая переменная, хранящаяся в памяти, с начальным значением

**Вход:** переменная, связанная с устройством ввода (обновляется системой)

**Выход:** переменная, связанная с устройством вывода

Замечание: Когда действительная переменная, привязана к устройству В/В, соответствующий драйвер В/В работает с эквивалентным целым значением.

Предупреждение: Целые и действительные аналоговые переменные или константы не могут смешиваться в одном и том же аналоговом выражении.

### V.2.3.5 Временные переменные

Временные означает **часы** и **счетчики**. Такие переменные могут принимать значения, выраженные в единицах времени и, обычно, используются во временных выражениях. Значение временной переменной не может превосходить **23h59m59s999ms** и не может быть отрицательным. Временная переменная хранится в 32 битовом слове. Его внутреннее представление - положительное число миллисекунд.

Временные переменные могут иметь один из следующих **атрибутов**:

**Внутренняя**: переменная, хранящаяся в памяти, изменяемая программой

**Константа**: неизменяемая переменная, хранящаяся в памяти, с начальным значением

**Предупреждение**: Временная переменная не может иметь атрибутов ВХОД или ВЫХОД.

Временные переменные автоматически обновляются системой ISaGRAF. Когда временная переменная активна, ее значение автоматически увеличивается в соответствии с часами реального времени целевой задачи. Для управления временными переменными используются следующие операторы языка **ST**:

**TSTART** запустить автоматическое обновление временной переменной (таймера)

**TSTOP** остановить автоматическое обновление временной переменной (таймера)

### V.2.3.6 Строковые переменные

Сообщения или строковые переменные содержат символьные строки. Длина строки может меняться в процессе работы. Длина строковой переменной не может превосходить значения определенного при объявлении переменной. Максимальная длина строки ограничена 255 символами.

Строковые переменные могут иметь один из следующих **атрибутов**:

**Внутренняя**: переменная, хранящаяся в памяти, изменяемая программой

**Константа**: неизменяемая переменная, хранящаяся в памяти, с начальным значением

**Вход**: переменная, связанная с устройством ввода (обновляется системой)

**Выход**: переменная, связанная с устройством вывода

Строка может содержать любой символ из стандартной таблицы ASCII (код ASCII от 0 до 255). Некоторые "C" функции из стандартной библиотеки ISaGRAF неправильно обрабатывают сообщения, содержащие нулевой символ (0).

## V.2.4 Комментарии

Комментарии могут быть свободно введены в текстовые языки, такие как **ST** и **IL**. Комментарии должны начинаться со специального символа "(" и заканчиваться символом "\*)". Комментарии могут быть введены в любом месте программы **ST** и могут занимать несколько строк.

Например:

```
counter := ivalue; (* присвоить значение основному счетчику *)
```

```
  (* это комментарий
```

```
    на двух строках *)
```

```
    c := counter (* комментарий можно расположить где угодно *) + base_value + 1;
```

Комментарии не должны перекрываться. Это означает, что символ "(" не может быть использован внутри комментария.



Предупреждение: Язык IL допускает комментарий в строке только после инструкции.

## B.2.5 Макроопределения

Система ISaGRAF позволяет переопределять константы, выражения true или false, ключевые слова и сложные выражения **ST**. Для этого соответствующему выражению должно быть дано имя **идентификатора**. Например:

```
YES      is      TRUE
PI       is      3.14159
OK       is      (auto_mode AND NOT (alarm))
```

Если такая эквивалентность определена, **идентификатор** может быть использован в любом месте **ST** программы вместо выражения. Например:

```
If OK Then
  angle := PI / 2.0;
  isdone := YES;
End_if;
```

Макроопределения могут быть локальными (**LOCAL**) для одной программы, глобальными (**GLOBAL**) или общими (**COMMON**).

Локальные макроопределения могут быть использованы только одной программой.

Глобальные макроопределения могут быть использованы любой программой в проекте.

Общие макроопределения могут быть использованы любой программой в любом проекте.

Заметим, что общие макроопределения могут быть сохранены отдельно менеджером архивов.

Предупреждение: Когда один и тот же идентификатор определяется в **ST** программе дважды, тогда используется последнее определение. Например:

```
определение :   ОТКРЫТЬ      is      FALSE
                 ОТКРЫТЬ      is      TRUE
```

означает: ОТКРЫТЬ is TRUE

Имена макроопределений должны удовлетворять следующим правилам:

- имя не может быть длиннее **16** символов

- первым символом должна быть **буква**

- последующими символами могут быть **буквы, цифры** или **символ подчеркивания**

Предупреждение: Макроопределение не может быть определено через другое макроопределение, например, нельзя написать:

```
PI       is      3.14159
PI2      is      PI*2
```

Пишите лучше:

```
PI2      is      6.28318
```

## В.3 Язык последовательных функциональных схем (SFC)

Язык последовательных функциональных схем (SFC) - это **графический** язык, который используется для описания **последовательных операций**. Процесс представляется в виде набора определенных **шагов**, связанных **переходами**. К каждому переходу прикреплено **логическое условие**. **Действия** внутри шагов описаны более детально при помощи других языков (**ST, IL, LD, FBD**).

### В.3.1 Основной формат схемы SFC

SFC программа - это графический набор **шагов** и **переходов**, соединенных вместе направленными связями. Для обозначения схождения и расхождений используются множественные связи. Некоторые части программы могут быть отделены и представлены в основной схеме одним символом - **макро шагом**. Вот основные **графические правила** для SFC:

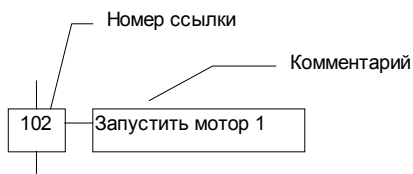
- шаги не могут следовать подряд
- переходы не могут следовать подряд

### В.3.2 Основные компоненты SFC

Основные компоненты SFC : шаги, начальные шаги, переходы, ориентированные связи, прыжки на шаг.

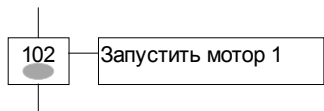
#### В.3.2.1 Шаги и начальные шаги

Шаг представляется одиночным **квадратом**. Каждому шагу присваивается **номер**, написанный внутри квадрата. Основное описание шага пишется внутри прямоугольника , присоединенного к символу шага. Это **свободный комментарий** (который не является частью языка). Вышеприведенная информация называется **Уровнем 1** шага:

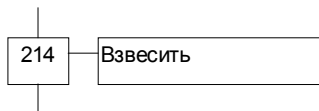


Во время работы **активный** шаг помечается **маркером**:

Активный шаг:

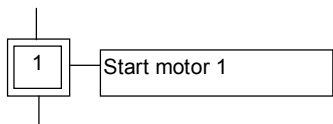


Неактивный шаг:



**Начальная ситуация** программы SFC описывается **начальными шагами**. Начальный шаг обозначается графическим символом с **двойной рамкой**. После запуска программы маркер автоматически устанавливается на каждый начальный шаг.

**начальный шаг:**



SFC программа должна содержать, по **крайней мере, один** начальный шаг.

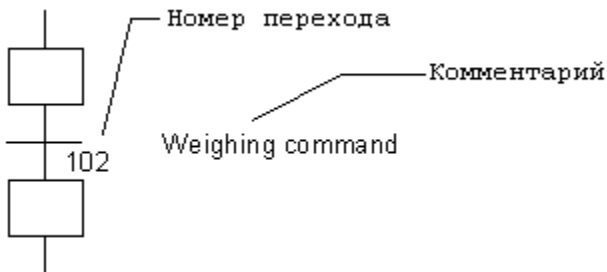
У шага есть атрибуты. Они могут быть использованы в любом другом языке:

**GSnnn.x**.....активность шага (логическая переменная)

**GSnnn.t**.....продолжительность активного состояния шага (таймер)  
(где nnn - номер начального шага)

### **В.3.2.2 Переходы**

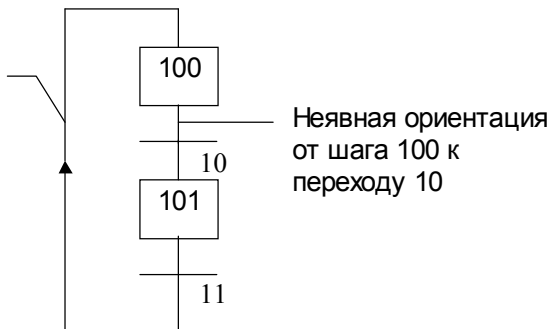
Переходы представлены горизонтальными полосками, которые пересекают линии связи. Каждому переходу присвоен номер, следующий за символом перехода. Описание перехода располагается справа от символа перехода. Это описание представляет собой **свободный комментарий** (не входящий в язык программирования). Вышеприведенная информация называется **Уровнем 1** перехода



### В.3.2.3 Ориентированные связи

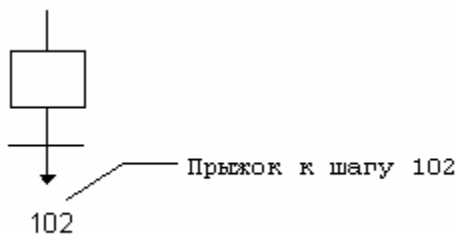
Для связи шагов и переходов используются одиночные линии. Это ориентированные связи. Когда ориентация не задана явно, связь ориентирована сверху вниз.

Явная ориентация от перехода 11 к шагу 100

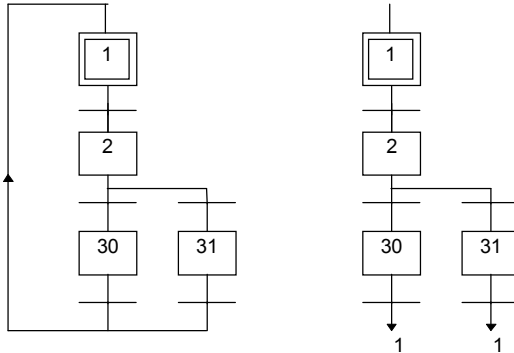


### В.3.2.4 Прыжок на шаг

Символ прыжка может быть использован, чтобы определить линию связи от перехода к шагу, не рисуя линию. Символ прыжка должен иметь номер шага назначения:



Связь от шага к переходу нельзя представить с помощью символа прыжка. Следующие схемы эквивалентны:

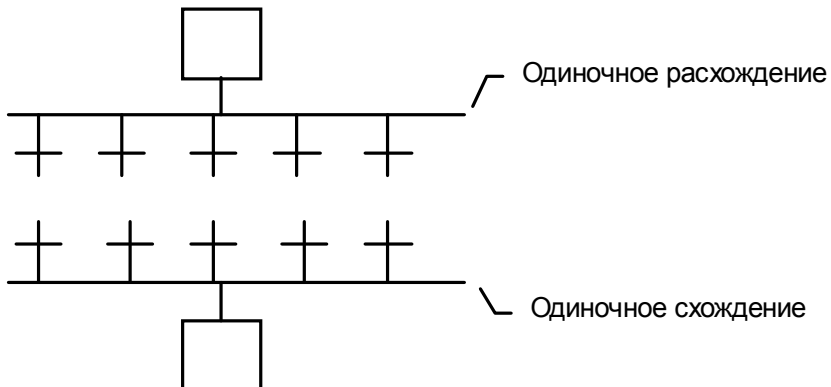


### В.3.3 Схождения и расхождения

Расхождения - это **множественные связи** от одного символа SFC (шага или перехода) ко многим. Схождение - это множественные связи от более чем одного символа SFC к одному другому символу. Схождения и расхождения могут быть одиночными или двойными.

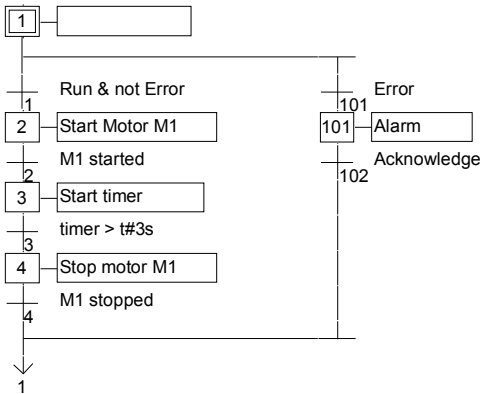
#### В.3.3.1 Одиночные расхождения

Одиночное расхождение - это множественная связь от одного шага к нескольким переходам. Оно позволяет маркеру активности пройти по одной из множества ветвей. Одиночное схождение - это множественная связь от нескольких переходов к одному и тому же шагу. Одиночное схождение, обычно используется для того, чтобы объединить несколько ветвей SFC, начавшихся из одиночного расхождения. Одиночные расхождения и схождения обозначаются одиночными горизонтальными линиями.



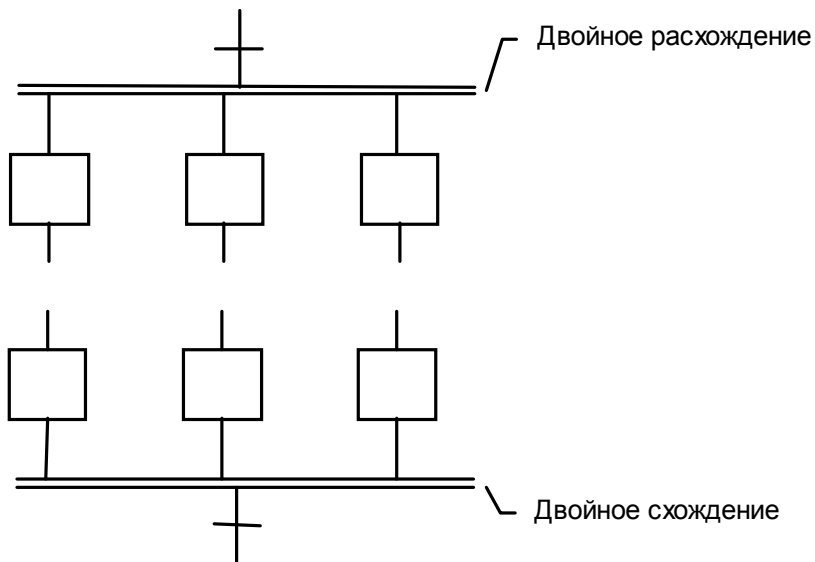
**Предупреждение:** Условия, присоединенные к переходам, не являются **взаимоисключающими**. Для того чтобы программа пошла по одной ветке, надо явно определить исключительность условий перехода. Например:

(\* Это программа SFC с одиночным схождением и расхождением\*)



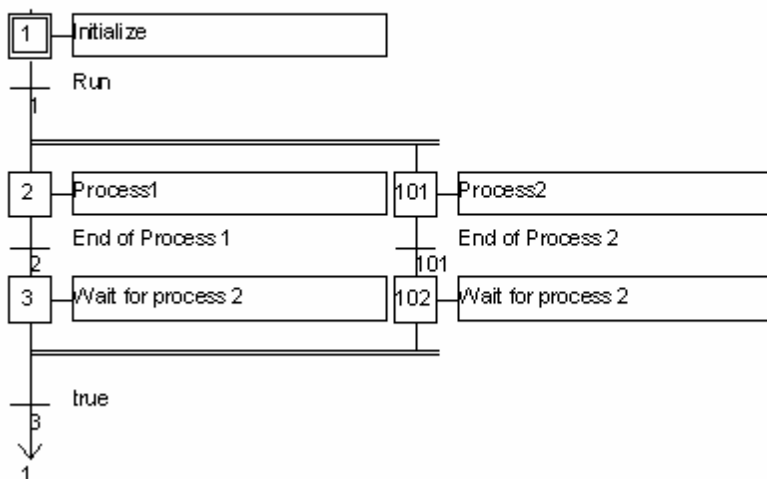
### В.3.3.2 Двойные расхождения

Двойное расхождение - это множественная связь от одного перехода к нескольким шагам. Она соответствует параллельной работе процесса. Двойное схождение - это множественная связь от нескольких шагов к одному и тому же переходу. Одиночное схождение, обычно, используется для того, чтобы объединить несколько ветвей SFC, начавшихся из двойного расхождения. Двойные расхождения и схождения обозначаются двойными горизонтальными линиями.



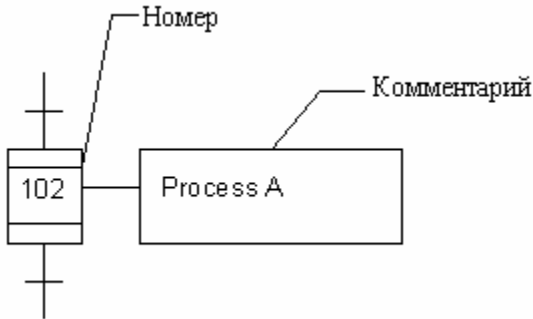
Пример двойного схождения и расхождения:

(\*SFC программа с двойными расхождениями и схождениями\*)



### В.3.4 Макро шаги

Макро шаг - это уникальное представление уникальной группы шагов и переходов. Тело макро шага описывается отдельно в другом месте SFC программы. В основной схеме SFC он выглядит как один символ. Для макро шага используется символ:



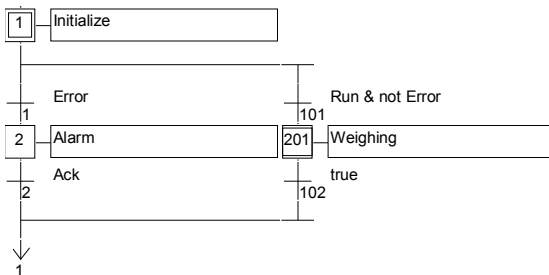
Номер, написанный в макро шаге, - это номер первого шага в теле макро шага. Тело макро шага должно начинаться **начальным шагом** и заканчиваться **конечным шагом**. Схема должна быть замкнутой. Начальный шаг не имеет верхней связи (нет перехода сзади). Конечный шаг не имеет нижней связи (нет перехода вперед). Символ макро шага может быть помещен в тело другого макро шага.

Предупреждение: Так как макро шаг это уникальный набор шагов и переходов, один и тот же макро шаг не может быть использован более чем однажды в SFC программе.

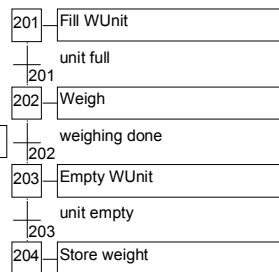
Пример:

(\* Программа SFC с макрошагом \*)

(\* Основная схема \*)



(\* Тело макрошага \*)





### В.3.5 Действия внутри шагов

Уровень 2 шага SFC представляет собой детальное описание **действий** в период **активности шага**. Это описание может использовать **текстовые дополнения языка SFC** и другие языки, такие как Структурный Текст (**ST**). Основные типы действий:

- Булевские действия
- Импульсные действия, запрограммированные на ST
- Не сохраняемые действия, запрограммированные на ST
- SFC действия

Несколько действий одинаковых или разных типов могут быть описаны в одном шаге. Средства, позволяющие использовать любой другой язык:

- Вызовы подпрограмм
- Соглашения языка инструкций (IL)

#### В.3.5.1 Булевские действия

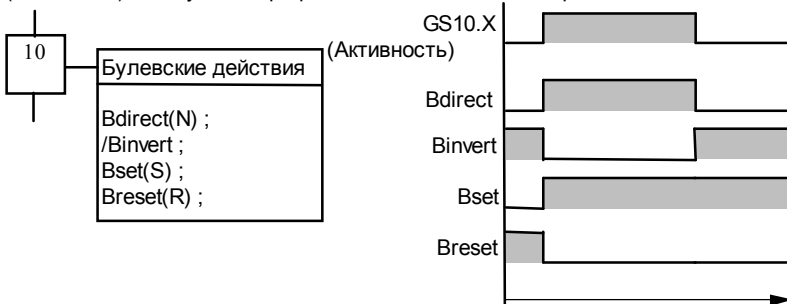
Булевские действия присваивают значение логической переменной при активизации шага. Логические переменные могут быть выходными или внутренними. Им присваивается значение каждый раз, когда шаг становится активным или перестает быть активным. Синтаксис основных логических действий:

**<boolean\_variable> (N) ;**                    присвоить переменной сигнал активности шага  
**<boolean\_variable> ;**                    тот же эффект (N не обязательно)  
**/ <boolean\_variable> ;**                    присвоить переменной отрицание сигнала активности шага

Есть другие возможности установки и сброса логических переменных, когда шаг становится активным. Синтаксис установки и сброса логических действий:

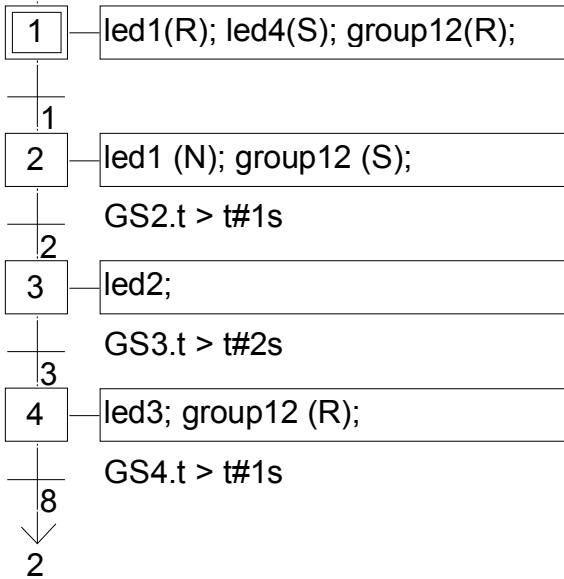
**<boolean\_variable> (S) ;**                    присваивает переменной значение TRUE, когда шаг  
**<boolean\_variable> (R) ;**                    присваивает переменной значение FALSE, когда шаг

Логические переменные должны быть выходными (OUTPUT) или внутренними (INTERNAL). Следующая программа ведет себя таким образом:



Пример:

(\* Программа SFC использующая булевские действия \*)

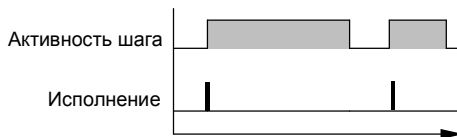


### В.3.5.2 Импульсные действия

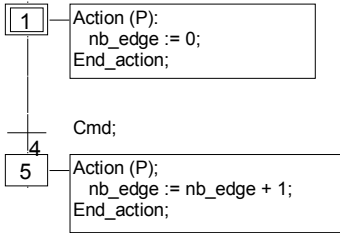
Импульсное действие - это список инструкций ST или IL, которые выполняются только **однажды** при **активизации** шага. Инструкции пишутся в соответствии со следующим синтаксисом:

**ACTION (P) :**  
(\*ST операторы \*)  
**END\_ACTION ;**

Вот результат импульсного действия:



Пример:



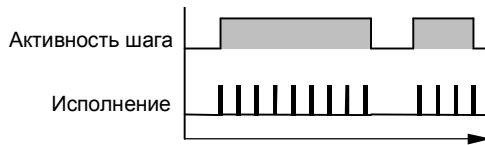
### V.3.5.3 Не сохраняемые действия

Не сохраняемое действие - это список инструкций ST или IL, которые выполняются **на каждом цикле**, в течение всего периода **активности** шага. Инструкции пишутся в соответствии со следующим синтаксисом:

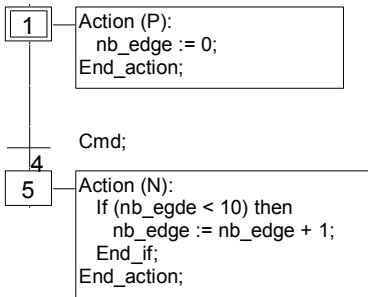
```

ACTION (N) :
  (* ST операторы *)
END_ACTION ;
  
```

Вот результат не сохраняемого действия:



Пример:



### V.3.5.4 Действия SFC

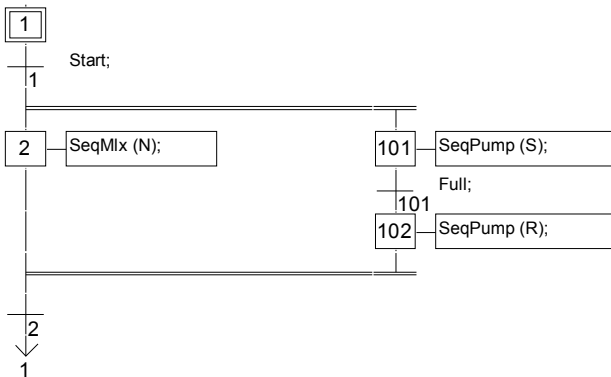
SFC действие - это дочерняя последовательность SFC, стартующая и убивающаяся в соответствии с изменением сигнала активности шага. SFC действие может иметь признак **N** (не запоминаемый), **R** (установить), **S** (сбросить). Вот синтаксис основных SFC действий:

<b>&lt;child_prog&gt; (N);</b>	запустить дочернюю последовательность, когда шаг становится активным и убить ее, когда шаг становится пассивным
<b>&lt;child_prog&gt; ;</b>	тот же эффект (N не обязательно)
<b>&lt;child_prog&gt; (S);</b>	запустить дочернюю последовательность, когда шаг становится активным и ничего не делать, когда шаг становится пассивным
<b>&lt;child_prog&gt; (R);</b>	убить дочернюю последовательность, когда шаг становится активным и ничего не делать, когда шаг становится пассивным

**SFC** последовательность, определенная как действие, должна быть **дочерней SFC программой** редактируемой программы. Заметим, что использование признаков **S** и **R** для **SFC** действия имеет тот же эффект, что и операторы **GSTART** и **GKILL** в импульсном действии на языке **ST**.

Ниже представлен пример SFC действия. Основная SFC программа называется **Father**. Она имеет два SFC наследника - **SeqMix** и **SeqPump**. Текст родительской SFC программы:

(\* Программа SFC использующая действия SFC \*)



**В.3.5.5 Вызов функции и функционального блока из действия**

Подпрограммы, функции или функциональные блоки (написанные на ST, IL, LD, или FBD) или "С" функции и "С" функциональные блоки, могут быть вызваны непосредственно из блока SFC действия на основе следующего синтаксиса:

Для подпрограмм и "С" функций:

```

ACTION (P) :
    result := sub_program ( ) ;
END_ACTION;
  
```

или

```
ACTION (N) :  
    result := sub_program ( ) ;  
END_ACTION;
```

Для функциональных блоков на "C" или на ST, IL, LD, FBD:

```
ACTION (P) :  
    Fbinst(in1, in2);  
    result1 := Fbinst.out1;  
    result2 := Fbinst.out2;  
END_ACTION;
```

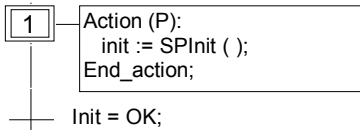
или

```
ACTION (N) :  
    Fbinst(in1, in2);  
    result1 := Fbinst.out1;  
    result2 := Fbinst.out2;  
END_ACTION;
```

Более подробный синтаксис можно найти в описании языка ST.

Пример вызова подпрограммы в блоке действия:

(\* Программа SFC с вызовом подпрограммы в блоке действия \*)



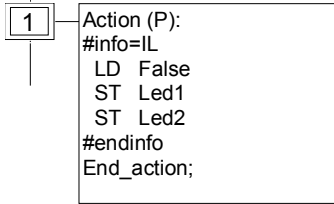
### В.3.5.6 Соглашения языка IL

Текст на языке IL может быть непосредственно введен в блок действия SFC, на основе следующего синтаксиса:

```
ACTION (P) :          (* или N *)  
#info=IL  
    <инструкция>  
    <инструкция>  
    ....  
#endinfo  
END_ACTION;
```

Специальные ключевые слова "#info=IL" и "#endinfo" должны быть введены именно так, прописными буквами. Пробелы и символы табуляции вводить нельзя до или после ключевых слов. Пример:

(\* SFC программ с последовательностью IL в блоке действия \*)



### В.3.6 Условия, присоединенные к переходам

К каждому переходу присоединяется **логическое выражение**, которое является условием прохождения этого перехода. Условие обычно записывается на языке ST или LD (быстрый LD редактор). Это **Уровень 2** перехода. Однако могут быть использованы и другие структуры:

- Соглашения языка ST
- Соглашения языка LD
- Соглашения языка IL
- Вызовы функций из переходов

Предупреждение: Если к переходу не присоединено выражение, то по умолчанию условие - TRUE.

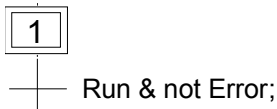
#### В.3.6.1 Соглашения языка ST

Язык **Структурный Текст** (ST) можно использовать для описания **условий**, присоединенных к переходам. Выражение должно иметь **логический** тип и заканчиваться точкой с запятой:

**< boolean\_expression > ;**

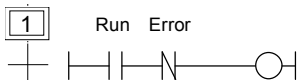
Выражение может быть константой TRUE или FALSE, входом или внутренней логической переменной, или комбинацией переменных, которые дают логическое значение. Пример:

(\* Программа SFC с программой ST для переходов \*)



#### В.3.6.2 Соглашения языка LD

Язык Релейных Диаграмм (LD) можно использовать для описания **условий**, присоединенных к переходам. Диаграмма состоит из штанги с витком. Значение витка представляет значение перехода. Пример:



### В.3.6.3 Соглашения языка IL

Язык **Список Инструкций** (IL) можно использовать для описания SFC переходов, согласно следующему синтаксису:

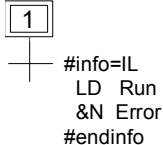
```
#info=IL
    <инструкция>
    <инструкция>
    ....
```

Значение, которое содержит **текущий результат** (IL регистр) в конце IL последовательности, будет являться условием присоединенным к переходу:

```
текущий результат = 0      è      условие FALSE
текущий результат <> 0    è      условие TRUE
```

Специальные ключевые слова "#info=IL" и "#endinfo" должны быть введены именно так, прописными буквами. Пробелы и символы табуляции вводить нельзя до или после ключевых слов. Пример:

(\* SFC программа с IL программой для переходов \*)



### В.3.6.4 Вызов подпрограмм из перехода

Любая подпрограмма или функция (написанные на FBD, LD, ST или IL) или "С" функция могут вычислять значение условия присоединенного к переходу, согласно следующему синтаксису:

```
< sub_program > ( ) ;
```

Значение, возвращаемое подпрограммой или функцией должно быть логическим и давать результирующее условие:

```
return value = FALSE      è      условие FALSE
return value = TRUE       è      условие TRUE
```

Пример:

(\* Программа SFC с вызовом программ для переходов \*)



— EvalCond ( );

### В.3.7 Динамические правила SFC

Вот пять динамических правил языка SFC:



#### Начальная ситуация

Начальная ситуация характеризуется начальными шагами, которые, по определению, находятся в активном состоянии в начале работы. По крайней мере, один начальный шаг должен быть в каждой SFC программе.



#### Освобождение перехода

Переход либо **разрешен**, либо **запрещен**. Говорят, что переход разрешен, если все непосредственно предшествующие шаги, связанные с соответствующим символом перехода, **активны**, в противном случае, переход запрещен. Переход не может быть освобожден, если :

- он не разрешен и
- соответствующее условие перехода не true.



#### Изменение состояния активного шага

Освобождение перехода одновременно ведет к активному состоянию непосредственно следующего шага и пассивному состоянию непосредственно предшествующего шага.



#### Одновременное освобождение переходов

Для того чтобы определить переходы, которые должны освобождаться одновременно, могут быть использованы двойные линии. Если такие переходы изображены отдельно, то активное состояние предшествующих шагов (GSnnn.x) может быть использовано, чтобы выразить их условия.



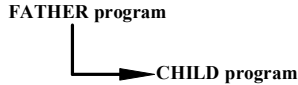
#### Одновременная активизация и дезактивация шага

Если во время работы шаг одновременно активизируется и дезактивируется, о приоритет отдается активизации.

### В.3.8 Иерархия программы SFC

Система ISaGRAF допускает описание вертикальной структуры программы SFC. SFC программы образуют **иерархическое дерево**. Каждая SFC программа может управлять (запускать, убивать) другие SFC программы. Такие программы называются наследниками SFC программы управляющей ими. SFC программы связаны в основное иерархическое дерево, используя связь **“родитель-наследник”**:





Основные правила иерархической структуры:

- **SFC** программы, которые не имеют родителей, называются "**основными**" SFC программами
- Основные **SFC** программы активизируются системой в момент запуска приложения
- Программа может иметь несколько программ-наследников
- Наследник не может иметь более одного родителя
- Программа-наследник управляется только своим родителем
- Программа не может управлять наследником своего наследника

Основные действия, которые SFC программа-родитель может выполнять для управления программой-наследником :

Start	Запустить <b>(GSTART)</b> Запускает дочернюю программу : активизирует каждый ее начальный шаг. Наследники этой программы не запускаются автоматически.
Kill	Убить <b>(GKILL)</b> Убивает дочернюю программу : деактивируя каждый ее активный шаг. Все наследники этой программы также убиваются.
Freeze	(Заморозить <b>(GFREEZE)</b> Деактивирует каждый активный шаг программы, и запоминает их так, чтобы программа могла быть перезапущена. Все наследники этой программы также замораживаются.
Restart	Перезапустить <b>(GRST)</b> Перезапускает замороженную дочернюю программу активизируя все ее зависшие шаги. Наследники этой программы не перезапускаются автоматически.

## В.4 Язык потоковых диаграмм

**Потоковые Диаграммы (FC)** – это графический язык, использующийся для описания **последовательных операций**. Потоковая Диаграмма состоит из **Действий** и **Тестов**. Между Действиями и Тестами находятся **ориентированные связи** представляющие потоки данных. Множественные связи используются для представления расхождений и схождений. Действия и тесты могут быть описаны с помощью языков ST, LD or IL. Функции и Функциональные блоки любого языка (кроме SFC) могут быть вызваны из действий и тестов. Программа Потоковых Диаграмм может вызывать другие программы Потоковых Диаграмм. Вызываемая программа FC – это **подпрограмма** вызывающей программы FC.

### В.4.1 Компоненты FC

Ниже даны компоненты языка Потоковых Диаграмм:

#### ▬ **Начало схемы FC**

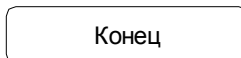
Символ "**начало**" должен появляться в начале программы Потоковых Диаграмм. Он уникален и не может быть опущен. Он представляет начальное состояние диаграммы, когда она активизирована. Ниже дан чертеж символа "начало" :



Символ "Начало" всегда имеет соединение (внизу) с другим объектом схемы. Потоковая диаграмма неверна, если нет соединения символа "Начало" с другим объектом.

#### ▬ **Конец схемы FC**

Символ "**конец**" должен возникать в конце программы Потоковых Диаграмм. Он уникален и не может быть пропущен. Может быть так, что никакого соединения не подходит к символу "Конец" (всегда виток), но символ "Конец" все же нарисован внизу схемы. Он представляет собой заключительное состояние схемы, когда исполнение было завершено. Ниже дан чертеж символа "конец":



Символ "Конец" обычно имеет соединение (наверху) с другими объектами схемы. Потоковая диаграмма может не иметь соединения с объектом "Конец" (бесконечный цикл). В этом случае объект "Конец" должен быть, тем не менее, виден внизу схемы.

#### ▬ **Потоковые связи FC**

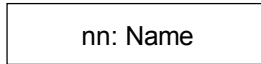
Потоковые **связи** – это линии, которые представляют потоки между двумя точками в диаграмме. Связь всегда заканчивается стрелкой. Ниже дан чертеж потоковой связи:



Две связи не могут исходить из одного источника.

### ▬ Действия FC

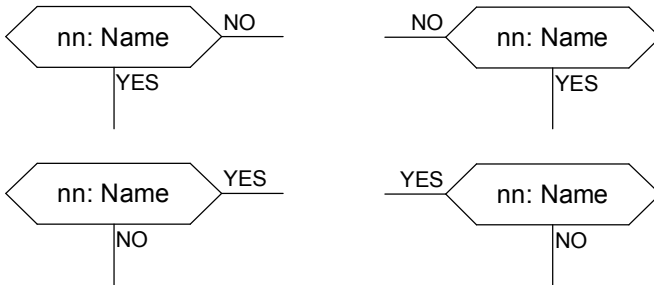
Символ **действия** представляет собой действие, которое нужно выполнить. Действия идентифицируются при помощи числа и имени. Ниже дан чертеж символа "действия" :



Два разных объекта одной схемы не могут иметь одно и то же имя или логический номер. Языками программирования для действий могут быть ST, LD или IL. Действия всегда соединены со связями, одна подходит к нему, другая исходит из него.

### ▬ Условия FC

**Условие** представляет собой булевский **тест**. Условие идентифицируется числом и именем. В соответствии со значением присоединенного выражения на ST, LD или IL, поток направляется либо по пути "YES", либо "NO". Ниже даны возможные чертежи символа условия:



Два различных объекта одной и той же схемы не могут иметь одно и тоже имя или логический номер. Программа теста

- выражение на ST, или
- одиночная ступень в LD, с символом присоединенном к уникальному витку, или
- несколько инструкций на IL. Регистр IL (или текущий результат) используется для того, чтобы оценить условие.

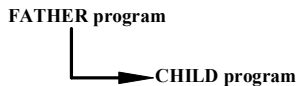
При программировании на ST, за выражение может следовать двоеточие. При программировании на LD, значение условия представляется уникальным витком. Условие равное:

- 0 or FALSE направляет поток по NO
- 1 or TRUE направляет поток по YES

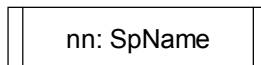
Тест всегда соединен с входящей связью, и оба выходящих соединения должны быть определены.

## ▬ Подпрограмма FC

Система допускает описание иерархической структуры программ FC. Программы FC организованы в виде **иерархического дерева**. Каждая программа FC может вызывать другие FC программы. Такая программа называется **дочерней программой** программы FC, которая ее вызывает. Программы FC, которые вызывают подпрограммы, называются **родительскими программами**. Программы FC объединяются вместе в общее иерархическое дерево, используя отношение "предок - наследник":



Символ **подпрограммы** в Поточковой Диаграмме представляет вызов подпрограммы. Исполнение вызывающей программы FC останавливается до завершения работы подпрограммы. Подпрограмма Поточковой Диаграммы идентифицируется числом и именем, как другие программы, функции или функциональные блоки. Ниже дан чертеж символа "вызова подпрограммы":



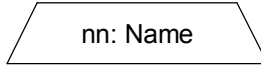
Два различных объекта одной и той же схемы не могут иметь одно и тоже имя или логический номер. Вот основные правила, по которым строится иерархическая структура FC:

- FC программы, которые не имеют родителей, называются основными FC программами.
- Основные программы FC активизируются системой при запуске приложения
- Программа может иметь несколько дочерних программ
- Дочерняя программа не может иметь более одного родителя
- Дочерняя программа может быть вызвана своим родителем
- Программа не может вызывать дочь своей собственной дочери

Одна и та же подпрограмма может возникать несколько раз в родительской схеме. Вызов подпрограммы Поточковой Диаграммы представляет собой полное исполнение подсхемы. Исполнение родительской схемы прекращается на время исполнения дочерней схемы. Блок вызова подпрограммы должен следовать тем же правилам соединений, что и действие.

## ▬ Специфические действия В/В FC

Символ **специфических действий В/В** представляет действие, которое должно быть выполнено. Как другие действия, специфические действия В/В идентифицируются номером и именем. Одна и та же семантика используется в стандартных действиях и специфических действиях В/В. Цель специфических действий В/В состоит только в том, чтобы сделать схему более читаемой и сфокусировать внимание на непереносимых частях схемы. Использование специфических действий В/В – дополнительная особенность. Ниже дан чертеж символа "специфических действий В/В":



Специфические блоки В/В ведут себя точно так же, как стандартные действия. Это относится к их свойствам, программированию на ST, LD или IL, и правилам соединения.

### ⇒ **Соединители FC**

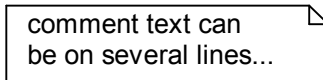
**Соединители** используются для представления связи между двумя точками диаграммы без вычерчивания. Соединитель обозначен как круг и связан с началом потока. Чертеж соединителя завершается, на соответствующей стороне (в зависимости от направления потока данных), с помощью идентификации точки цели (обычно имя символа цели). Ниже дан стандартный чертеж соединителя:



Соединитель всегда попадает в определенный в Поточковой Диаграмме символ. Символ назначения определяется его логическим номером.

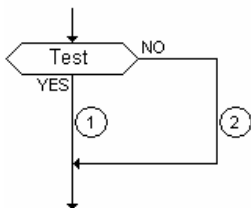
### ⇒ **Комментарии FC**

Блок **комментариев** содержит текст, который не имеет смысла для семантики схемы. Он может быть введен на любом свободном месте в окне Поточковой Диаграммы, и используется для документирования программ. Ниже дан чертеж символа "комментария":



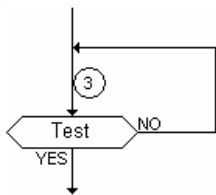
## **В.4.2 Сложные структуры FC**

Эта секция показывает примеры **сложных структур**, которые могут быть определены в Поточковых Диаграммах. Такие структуры объединяют основные объекты связанные вместе.



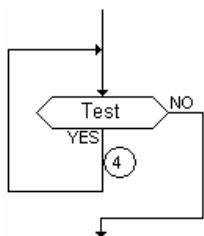
### **IF / THEN / ELSE**

- (1) место для ввода действий по "THEN"
- (2) место для ввода действий по "ELSE"



### REPEAT / UNTIL

(3) место для ввода повторяемых действий



### WHILE / DO

(3) место для ввода повторяемых действий

## В.4.3 Динамическое поведение FC

**Исполнение** Поточковой Диаграммы может быть представлено следующим образом:

- Символ Begin занимает один целевой цикл.
- Символ Конец занимает один целевой цикл и заканчивает исполнение схемы. После того, как этот символ достигнут, никаких действий на схеме не выполняется.
- Исполнение действия занимает один целевой цикл.

Замечание: В противоположность SFC, действие не постоянное состояние. Инструкции не повторяются, пока символ выделен.

## В.4.4 Проверка FC

Кроме присоединенных ST, LD или IL программ, некоторые другие **синтаксические правила** применяются к потоковой диаграмме. Ниже дан список основных правил:

- Все точки "соединения" всех символов должны быть связаны. (связь с символом "Конец" может быть опущена)
- Все символы должны быть связаны (не должно быть изолированных частей)
- Все соединители должны иметь правильный пункт назначения

Могут возникать другие синтаксические ошибки:

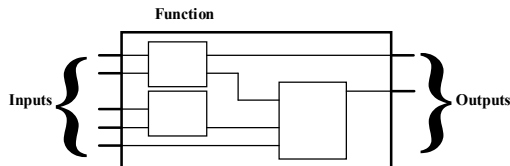
- Пустые действия (нет программы) рассматриваются, как шаги во время работы
- Пустые тесты (нет программы) рассматриваются, как "всегда истинные"

## V.5 Язык функциональных блочных диаграмм (FBD)

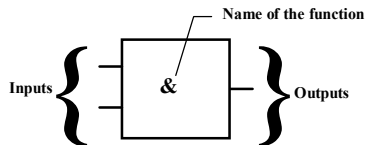
Язык **функциональных блочных диаграмм (FBD)** - графический язык. Он позволяет программисту строить сложные процедуры, используя существующие **функции** из библиотеки ISaGRAF и **связывая** их вместе при помощи графических диаграмм.

### V.5.1 Основной формат диаграмм FBD

FBD диаграмма описывает функцию между **входными переменными** и **выходными переменными**. Функция описывается как множество **элементарных функциональных блоков**. Входные и выходные переменные связываются в блоки при помощи линий связи. Выход функционального блока может быть также связан с входом другого функционального блока.



Вся функция FBD программы построена из стандартных **элементарных** функциональных блоков из библиотеки ISaGRAF. Каждый функциональный блок имеет фиксированное **количество входных точек связи** и фиксированное количество **выходных точек связи**. Функциональный блок представляется одиночным **прямоугольником**. Входы соединяются с **левым** краем. Выходы соединяются с **правым** краем. Элементарный функциональный блок реализует одну функцию между входами и выходами. Имя функции, реализуемой блоком, пишется на символе прямоугольника. Каждый вход или выход блока имеют определенный тип.



Входные переменные FBD программы должны быть связаны с точками входа функционального блока. Тип каждой переменной должен быть тем же что и тип соответствующего входа. Входом FBD блока может быть **константа**, любая **внутренняя**, **входная** или **выходная** переменная.

Выходные переменные FBD программы должны быть связаны с точками выхода функционального блока. Тип каждой переменной должен быть тем же что и тип соответствующего выхода. Выходом FBD блока может быть внутренняя или выходная переменная или имя программы (только для подпрограмм). Когда выходом является имя редактируемой подпрограммы, оно представляет присвоение возвращаемого значения подпрограммы (возвращаемого в вызывающую программу). Входные и выходные переменные, входы и выходы функциональных блоков соединены линиями связи. Линии могут быть использованы для соединения двух логических точек диаграммы:

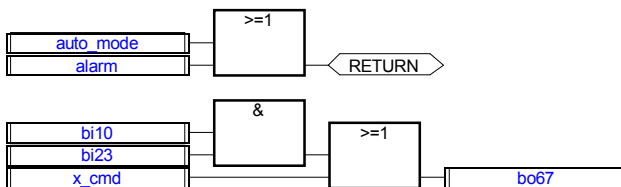
- Входной переменной и входа функционального блока
- Выхода функционального блока и входа другого блока
- Выхода функционального блока и выходной переменной

Связи ориентированы, это означает, что данные передаются с левого конца к правому. Левый и правый концы связи должны быть **одного типа**. Множественные правые связи могут быть использованы чтобы передать информацию от его левого конца к каждому правому концу. Все концы связи должны быть одного типа.

## B.5.2 Оператор RETURN

Ключевое слово **<RETURN >** может быть выходом диаграммы. Оно должно быть связано с логическим выходом функционального блока. Оператор **<RETURN >** представляет собой условное завершение программы: если выход блока связанного с оператором имеет тип **TRUE**, остальная часть диаграммы не выполняется.

(\* Пример FBD программы использующей оператор RETURN \*)



(\* ST эквивалент: \*)

```
If auto_mode OR alarm Then
    Return;
End_if;
bo67 := (bi10 AND bi23) OR x_cmd;
```

## B.5.3 Прыжки и метки

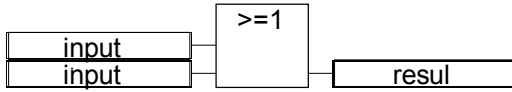
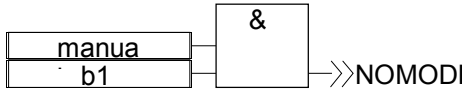
Прыжки и метки используются для управления выполнением диаграммы. К правому краю символа метки или прыжка не может быть присоединено никаких других объектов. Используются следующие обозначения:

**>>LAB**..... прыжок на метку (имя метки "LAB")  
**LAB**:..... определение метки (имя метки "LAB")

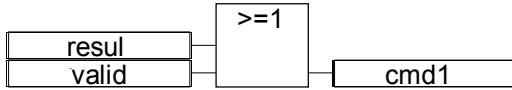
Если линия связи слева от символа прыжка находится в состоянии **TRUE**, исполнение программы переходит на соответствующую метку.

(\* Пример FBD программы с использованием меток и прыжков \*)





NOMODI



(\* IL эквивалент: \*)

```

ld      manual
and     b1
jmpc   NOMODIF
ld      input1
or      input2
st      result
NOMODIF:
ld      result
or      valid
st      cmd10

```

### В.5.4 Логическое отрицание

Одиночная линия связи с правым концом, присоединенным к входу функционального блока, может заканчиваться **логическим отрицанием**. Отрицание представляется маленьким колечком. Когда используется логическое отрицание, левый и правый концы линии связи должны иметь тип **BOOLEAN**.

(\* Пример FBD программы использующей метки и прыжки \*)



(\* ST эквивалент: \*)

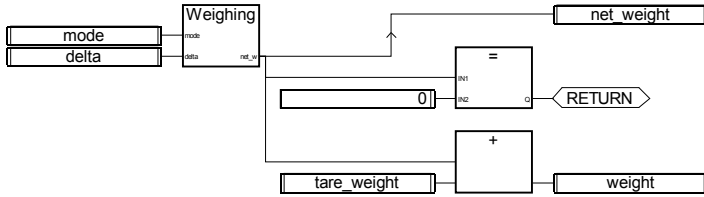
```
output1 := input1 AND NOT (input2);
```

### В.5.5 Вызов функций и функциональных блоков из FBD

Язык FBD позволяет вызывать подпрограммы, функции и функциональные блоки. Подпрограммы, функции и функциональные блоки представляются прямоугольником функции. Имя, написанное в прямоугольнике, - это имя подпрограммы, функции или функционального блока. В случае подпрограммы или функции единственным выходом

прямоугольника является возвращаемая величина. Функциональные блоки могут иметь более одного выхода.

(\* Пример FBD программы использующей блок SUB PROGRAM \*)



(\* ST Эквивалент \*)

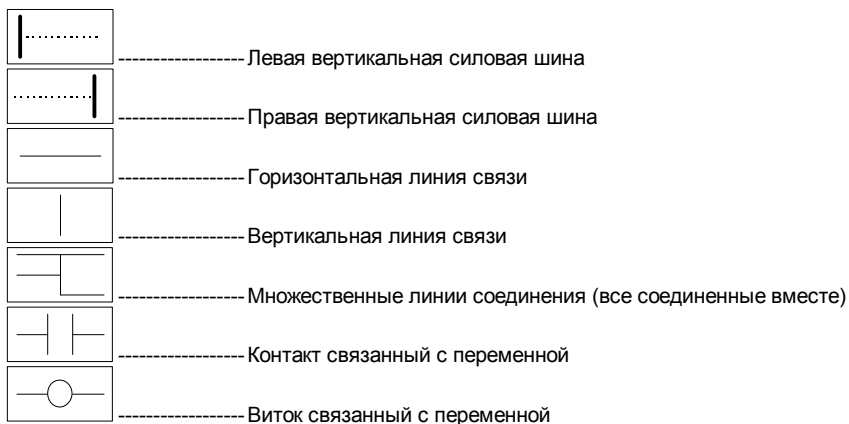
net\_weight := Weighing (mode, delta); (\*вызов подпрограммы\*)

If (net\_weight = 0) Then Return; End\_if;

weight := net\_weight + tare\_weight;

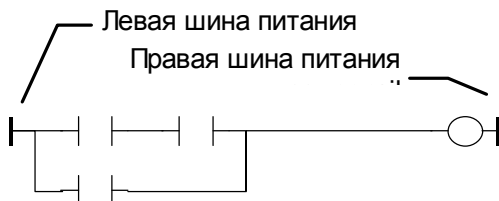
## В.6 Язык релейных диаграмм (LD)

Язык релейных диаграмм - это графическое представление логических уравнений, комбинирующее **контакты** (входы) и **витки** (выходы). Язык LD позволяет описывать работу с **булевыми** данными помещая **графические символы** в схему программы. Графические символы LD организованы внутри схемы так же, как электрическая схема. Справа и слева LD диаграмма должна соединяться с вертикальными силовыми рельсами. Основные компоненты LD диаграммы.

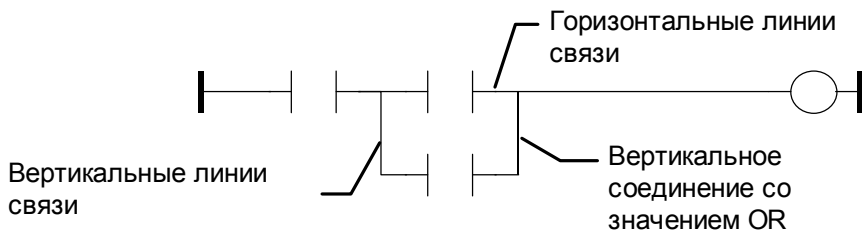


### В.6.1 Силовые рельсы и соединительные линии

LD диаграмма ограничена справа и слева вертикальными линиями, которые называются **левым силовым рельсом** и **правым силовым рельсом** соответственно.



Символы LD диаграммы связаны с силовыми рельсами и другими символами при помощи соединительных линий. Соединительные линии могут быть горизонтальными или вертикальными.



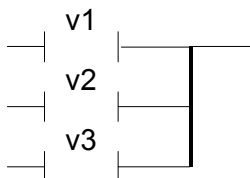
Каждый отрезок линии имеет состояние **TRUE** или **FALSE**. Отрезки, соединенные напрямую имеют одно и то же булевское состояние. Любая горизонтальная линия, соединенная с **левым вертикальным рельсом** имеет состояние **TRUE**.

### В.6.2 Множественные соединения

Левый и правый концы соединительной линии имеют одно и тоже логическое состояние. Комбинируя вертикальные и горизонтальные линии можно строить **множественные соединения**. Состояния концов множественного соединения определяются правилами логики.

**Левое множественное соединение** объединяет **более чем одну** горизонтальную линию, соединенную с вертикальной линией и одну горизонтальную линию подходящую с **правой** стороны. Булевское состояние правого конца - это **логическое ИЛИ(OR)** всех левых концов.

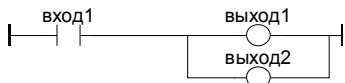
(\* Пример множественного левого соединения\*)



(\* состояние правого конца (v1 OR v2 OR v3) \*)

**Правое множественное соединение** объединяет **одну** горизонтальную линию, соединенную с вертикальной линией с **левой** стороны и **более чем одну** горизонтальную линию подходящую с **правой** стороны. Булевское состояние левого конца распространяется на все правые концы.

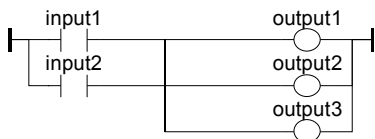
(\* Пример множественного ПРАВОГО соединения\*)



(\* ST эквивалент: \*)  
 output1 := input1;  
 output2 := input1;

**Множественное соединение слева и справа** объединяет **более чем одну** горизонтальную линию соединенную с вертикальной линией с левой стороны и **более чем одну** горизонтальную линию подходящую с правой стороны. Булевское состояние каждого правого конца - это логическое ИЛИ(OR) всех левых концов.

(\*Пример множественного ЛЕВОГО и ПРАВОГО соединения \*)



(\* ST Эквивалент: \*)  
 output1 := input1 OR input2;  
 output2 := input1 OR input2;  
 output3 := input1 OR input2;

### В.6.3 Основные контакты и витки языка LD

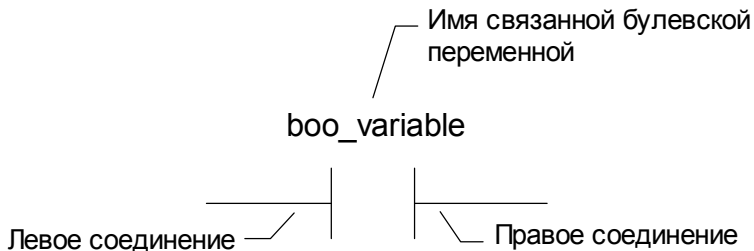
Для представления контактов используются символы:

- Прямой контакт
- Инvertированный контакт
- Контакт с определением фронта

Для представления витков используются символы:

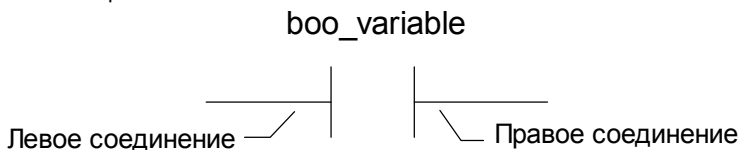
- Прямой виток
- Инvertированный виток
- SET виток
- RESET виток
- Виток с определением фронта

Имя переменной пишется над этими графическими символами:



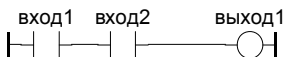
### ▬ **Прямой контакт**

Прямой контакт позволяет производить логические операции между состояниями линий и логическими переменными.



Состояние линии соединения на правом конце - это логическое И(AND) состояния левого конца и значения переменной контакта.

(\* Пример использования ПРЯМЫХ \*)

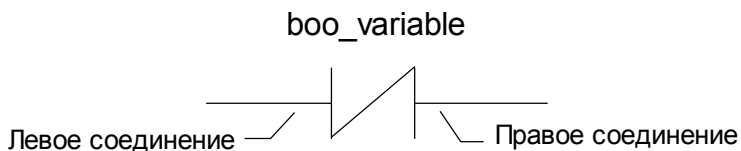


(\* ST Эквивалент: \*)

output1 := input1 AND input2;

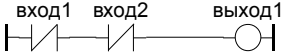
### ▬ **Инвертированный контакт**

Инвертированный контакт позволяет производить логические операции между состоянием линии и отрицанием логической переменной.



Состояние линии соединения на правом конце - это логическое И(AND) состояния левого конца и отрицания значения переменной контакта.

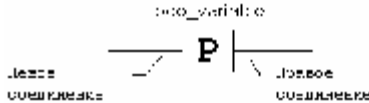
(\* Пример использования ИНВЕРТИРОВАННЫХ контактов\*)



(\* ST Эквивалент: \*)  
`output1 := NOT (input1) AND NOT (input2);`

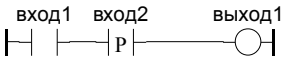
### ☰ **Контакт с определением переднего фронта**

Этот контакт позволяет производить логические операции между состоянием линии и логической переменной.



Состояние линии соединения на правом конце принимает значение TRUE, когда значение на левом конце - TRUE и значения переменной контакта меняется с FALSE на TRUE. Во всех остальных случаях оно устанавливается равным FALSE.

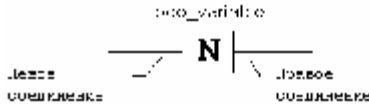
(\* Пример использования контактов ПЕРЕДНЕГО ФРОНТА\*)



(\* ST Эквивалент: \*)  
`output1 := input1 AND (input2 AND NOT (input2prev));`  
 (\* input2prev - значение input2 на предыдущем цикле \*)

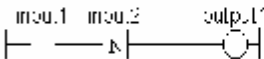
### ☰ **Контакт с определением заднего фронта**

Этот контакт позволяет производить логические операции между состоянием линии и логической переменной.



Состояние линии соединения на правом конце принимает значение TRUE когда значение на левом конце - TRUE и значения переменной контакта меняется с TRUE на FALSE. Во всех остальных случаях оно устанавливается равным FALSE.

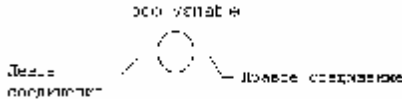
(\* Пример использования контактов ЗАДНЕГО ФРОНТА\*)



(\* ST Эквивалент: \*)  
 output1 := input1 AND (NOT (input2) AND input2prev);  
 (\* input2prev - значение input2 на предыдущем цикле \*)

### ⇒ **Прямой виток**

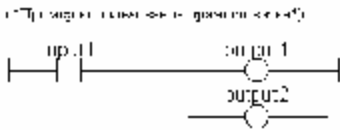
Прямой виток дает логический выход состояния линии соединения.



Переменной витка присписывается значение состояния левой линии соединения. Состояние левого соединения распространяется на правое соединение. Правое соединение может быть связано с правым вертикальным силовым рельсом.

Соответствующая логическая переменная должна быть выходом или внутренней.

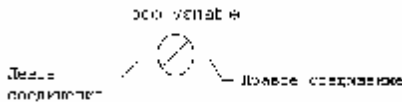
Соответствующее имя может быть именем программы (только для подпрограмм). Это соответствует присвоению возвращаемой величины подпрограммы.



(\* ST Эквивалент: \*)  
 output1 := input1;  
 output2 := input1;

### ⇒ **Инвертированный виток**

Инвертированный виток дает логический выход отрицания состояния линии соединения.



Переменной витка присписывается отрицание значение состояния левой линии соединения. Состояние левого соединения распространяется на правое соединение. Правое соединение может быть связано с правым вертикальным силовым рельсом.

Соответствующая логическая переменная должна быть выходом или внутренней.

Соответствующее имя может быть именем программы (только для подпрограмм). Это соответствует присвоению возвращаемой величины подпрограммы.



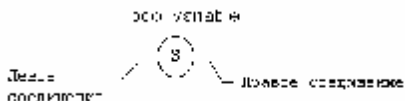
(\*Пример: установка значения инверсного бита\*)



(\* ST Эквивалент: \*)  
 output1 := NOT (input1);  
 output2 := input1;

### ▣ SET виток

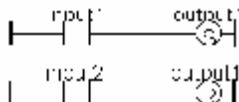
SET виток дает **логический выход** состояния линии соединения.



Соответствующая переменная принимает значение TRUE, когда **состояние левой линии связи** становится равным TRUE. Переменная удерживает это состояние до тех пор, пока она не будет инвертирована витком RESET. Состояние левого соединения распространяется на правое соединение. Правое соединение может быть связано с правым вертикальным силовым рельсом.

Соответствующая логическая переменная должна быть выходной или внутренней.

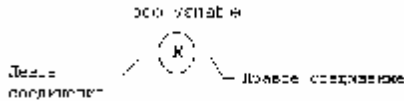
(\*Пример: установка значения бита SET и RESET\*)



(\* ST Эквивалент: \*)  
 IF input1 THEN  
 output1 := TRUE;  
 END\_IF;  
 IF input2 THEN  
 output1 := FALSE;  
 END\_IF;

### ▣ RESET виток

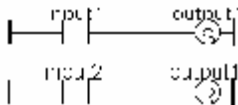
RESET виток дает логический выход состояния линии соединения.



Соответствующая переменная принимает значение FALSE, когда состояние левой линии связи становится равным TRUE. Переменная удерживает это состояние до тех пор, пока она не будет инвертирована витком SET. Состояние левого соединения распространяется на правое соединение. Правое соединение может быть связано с правым вертикальным силовым рельсом.

Соответствующая логическая переменная должна быть выходной или внутренней.

(\*Пример использования витка SET в Ladder\*)

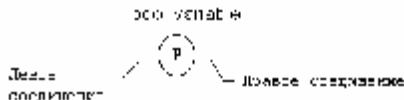


(\* ST Эквивалент: \*)

```
IF input1 THEN
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;
```

### **Виток с определением переднего фронта**

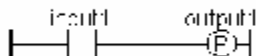
“Положительный” виток дает логический выход состояния линии соединения. Этот тип витка имеется только в “быстром” релейном редакторе (Quick ladder).



Соответствующая переменная принимает значение TRUE, когда состояние левой линии связи меняет значение с FALSE на TRUE. Во всех остальных случаях переменная принимает значение FALSE. Состояние левого соединения распространяется на правое соединение. Правое соединение может быть связано с правым вертикальным силовым рельсом.

Соответствующая логическая переменная должна быть выходной или внутренней.

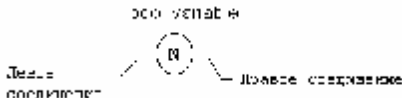
←T) 4444 44 4444 444 - 4 44 44444444 - 444444



```
(* ST Эквивалент: *)
IF (input1 and NOT(input1prev)) THEN
  output1 := TRUE;
ELSE
  output1 := FALSE;
END_IF;
(* input1prev - значение input1 на предыдущем цикле *)
```

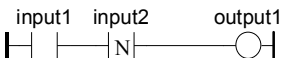
### Виток с определением заднего фронта

“Отрицательный” виток дает логический выход состояния линии соединения. Этот тип витка имеется только в “быстром” релейном редакторе (Quick ladder).



Соответствующая переменная принимает значение TRUE, когда состояние левой линии связи меняет значение с TRUE на FALSE. Во всех остальных случаях переменная принимает значение FALSE. Состояние левого соединения распространяется на правое соединение. Правое соединение может быть связано с правой вертикальным силовым рельсом.  
Соответствующая логическая переменная должна быть выходной или внутренней.

(\* Пример использования “Отрицательного” витка \*)



```
(* ST Эквивалент: *)
IF (NOT(input1) and input1prev) THEN
  output1 := TRUE;
ELSE
  output1 := FALSE;
END_IF;
(* input2prev - значение input2 на предыдущем цикле *)
```

### В.6.4 Оператор RETURN

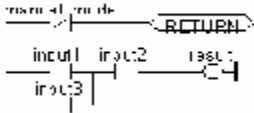
Метка RETURN может быть использована как выход, чтобы представить условное завершение программы. Никаких символов к правому концу RETURN подключать нельзя.

## RETURN

Если левый конец линии соединения имеет состояние TRUE, то программы завершается, не выполняя уравнения введенные в следующих строках программы.

**Замечание:** Если LD программа - это подпрограмма, то ее имя должно быть связано с выходным витком, чтобы установить возвращаемое значение.

(\* Пример использования оператора RETURN \*)



(\* ST Эквивалент: \*)

```
If Not (manual_mode) Then RETURN; End_if;
result := (input1 OR input3) AND input2;
```

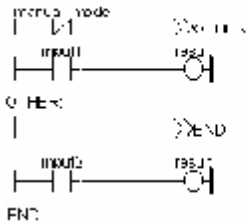
### B.6.5 Прыжки и метки

Метки, условные и безусловные прыжки, могут быть использованы для управления выполнением диаграммы. Никаких символов к правому концу символа метки и прыжка подключать нельзя. Используются следующие обозначения:

**>>LAB**..... прыжок на метку "LAB"  
**LAB**:..... определение метки "LAB"

Если линия связи слева от символа прыжка находится в состоянии **TRUE**, исполнение программы переходит на соответствующую метку.

(\* Пример использования операторов прыжка и метки \*)



(\* IL Эквивалент: \*)

ldn	manual_mode
jmpc	other
ld	input1
st	result

```

OTHER:      jmp      END
            ld       input2
            st       result
END:        (* конец программы *)

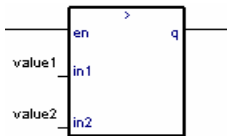
```

## В.6.6 Блоки в LD

Используя Быстрый LD редактор, вы подключаете функциональные блоки к логическим линиям. Функция, в действительности, может быть оператором, функциональным блоком или функцией. Так как блоки не всегда имеют логические входы и/или логические выходы, введение блоков в LD диаграммы приводит к добавлению нескольких новых параметров EN, ENO в интерфейс блока. Параметры EN, ENO не добавляются, если вы используете FBD/LD редактор, и вы можете подключить переменные требуемого типа.

### Вход "EN"

В некоторых операторах, функциях или функциональных блоках первый вход не булевский. Так как первый вход всегда должен быть подключен к шине, на первую позицию автоматически вводится другой вход, называемый "EN". Блок выполняется только тогда, когда вход EN равен TRUE. Ниже представлен пример оператора сравнения и эквивалентный код на ST:



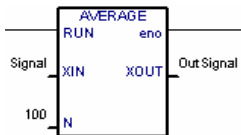
```

IF rung_state THEN
    q := (value1 > value 2);
ELSE
    q := FALSE;
END_IF;
(* продолжить шину с состоянием q *)

```

### Выход "ENO"

В некоторых операторах, функциях или функциональных блоках первый выход не булевский. Так как первый выход всегда должен быть подключен к шине, на первую позицию автоматически вводится другой выход, называемый "ENO". Выход ENO всегда имеет то же значение, что и первый вход блока. Ниже представлен пример функционального блока AVERAGE и эквивалентный код на ST:



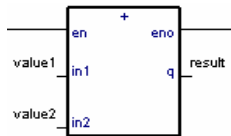
```

AVERAGE(rung_state, Signal, 100);
OutSignal := AVERAGE.XOUT;
eno := rung_state;
(* продолжить шину с состоянием eno*)

```

### параметры "EN" и "ENO"

В некоторых случаях требуются EN и ENO. Ниже представлен пример с арифметическим оператором и эквивалентный код на ST:



```
IF rung_state THEN
    result := (value1 + value2);
END_IF;
eno := rung_state;
(* продолжить шину с состоянием eno *)
```

## В.7 Язык структурированный текст (ST)

**ST (Structured text)** - это структурный язык высокого уровня разработанный для процессов автоматизации. Этот язык, в основном, используется для создания сложных процедур, которые не могут быть легко выражены при помощи графических языков. По умолчанию ST является языком для описания действий внутри шагов и условий языка **SFC**.

### В.7.1 Основной синтаксис ST

ST программа - это список **ST операторов**. Каждый оператор заканчивается точкой с запятой (;). Имена используемые в исходном коде (идентификаторы переменных, константы, ключевые слова) разделены **неактивными разделителями** (пробелами, символами окончания строки и табуляции) или **активными разделителями**, которые имеют определенное значение (например, разделитель ">" означает сравнение "больше чем". В текст могут быть введены комментарии. Комментарий должен начинаться с "(" и заканчиваться ")". Каждый оператор заканчивается точкой с запятой (";"). Основные операторы языка ST:

- оператор **присвоения** (variable := expression)
- вызов **подпрограммы** или **функции**
- вызов **функционального блока**
- операторы **выбора** (IF, THEN, ELSE, CASE)
- **итеративные** операторы (FOR, WHILE, REPEAT)
- **управляющие** операторы (RETURN, EXIT)
- специальные операторы для связи с такими языками как **SFC**

Неактивные разделители могут быть свободно введены между активными разделителями, константами и идентификаторами. Неактивные разделители - это **пробелы**, **символы окончания строки** и **табуляции**. В отличие от неформатных языков, таких как IL конец строки может быть введен в любом месте программы. Для улучшения читаемости программ нужно использовать неактивные разделители в соответствии со следующими правилами:

- Не пишите более одного оператора в строке
- Используйте табуляцию для сдвига сложных операторов
- Вводите комментарии для улучшения читаемости строк и параграфов

### В.7.2 Выражения и скобки

Выражения ST объединяют **операторы** и **операнды** (переменные или константы). Для одного выражения (объединяющего операнды с одним ST оператором), **тип** операндов должен быть одним и тем же. Это одно выражение имеет тот же тип, что и его операторы и может быть использовано в более сложном выражении. Например:

(boo_var1 AND boo_var2)	BOO тип
not (boo_var1)	BOO тип
(sin (3.14) + 0.72)	REAL ANALOG тип

(t#1s23 + 1.78)

неправильное выражение

**Скобки** используются для того чтобы отделить подчасти выражения и явно определить приоритетность операций. Когда в сложном выражении нет скобок, приоритетность ST операторов задана неявно. Например:

2 + 3 \* 6

равно 2+18=20

оператор \* имеет высший приоритет

(2+3) \* 6

равно 5\*6=30

приоритет задается скобками

Предупреждение: Максимальное количество вложенных скобок - 8.

### В.7.3 Вызовы функций и функциональных блоков

Стандартные ST вызовы функций могут быть использованы для каждого из следующих объектов:

- Подпрограммы
- Библиотечные функции и функциональные блоки, написанные на IEC языках
- "C" функции и функциональные блоки
- Функции преобразования типов

#### ▬ **Вызовы подпрограмм или функций**

**Имя:** имя вызываемой подпрограммы или библиотечной функции написанной на IEC языке или "C"

**Значение:** вызывает ST, IL, LD, или FBD подпрограммы или функции или "C" функции и дает возвращаемое значение

**Синтаксис:** <variable> := <subprog> (<par1>,...<parN>);

**Операнды:** За типом возвращаемого значения и параметрами вызова должен следовать интерфейс определенный для подпрограммы

**Возвращаемое значение:** значение возвращаемое подпрограммой

Вызовы подпрограмм могут быть использованы в выражении. Они также могут быть использованы в переходах SFC.

Пример 1: вызов подпрограммы

(\* Основная ST программа \*)

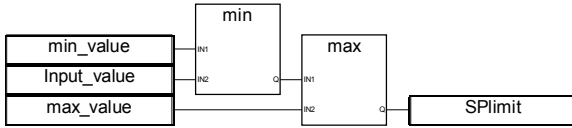
(\* получает аналоговое значение и превращает его во временное значение \*)

ana\_timeprog := SPLimit ( tprog\_cmd );

appl\_timer := tmr (ana\_timeprog \* 100);

(\* Вызывается FBD программа 'SPLimit' \*)





Пример 2: Вызов функции

(\* функции использующиеся в сложных выражениях: min, max, right, mlen and left - стандартные "С" функции \*)

```

limited_value := min (16, max (0, input_value) );
rol_msg := right (message, mlen (message) - 1) + left (message, 1);
  
```

## == Вызов функциональных блоков

**Имя:** имя экземпляра функционального блока  
**Значение:** вызывает функциональный блок из библиотеки ISaGRAF или библиотеки пользователя и передает возвращаемые параметры

**Синтаксис:** (\* вызвать функциональный блок \*)  
 <blockname> ( <p1>, <p2> ... );  
 (получить возвращаемые параметры \*)  
 <result> := <blockname>. <ret\_param1>;  
 ...  
 <result> := <blockname>. <ret\_paramN>;

**Операнды:** Параметры и выражения, которые имеют тот же тип, что и параметры этого функционального блока

**Возвращаемое значение:** См. Синтаксис получения возвращаемых параметров

Смотрите библиотеку ISaGRAF для того, чтобы найти значение и тип каждого параметра функционального блока. Экземпляр функционального блока (имя копии) должно быть объявлено в словаре.

Пример:

(\*вызов функционального блока ST программой \*)

(\* объявить экземпляр функционального блока в словаре: \*)

(\* trigb1 : block R\_TRIG - определение переднего фронта\*)

(\* активизация функционального блока из языка ST \*)

trigb1 (b1);

(\* доступ к возвращаемым параметрам \*)

If (trigb1.Q) Then nb\_edge := nb\_edge + 1; End\_if;

### B.7.4 Специфические булевские операторы языка ST

Следующие логические операторы являются специфическими для языка ST:

- REDGE определение восходящего фронта
- FEDGE определение падающего фронта

Могут быть использованы другие логические операторы, такие, как:

- NOT логическое отрицание
- AND(&) логическое И (AND)
- OR логическое ИЛИ (OR)
- XOR логическое исключающее ИЛИ (OR)

Их описание можно найти в главе 'Стандартные операторы функциональные блоки и функции'.

### ═ **"REDGE" оператор**

**Имя:** REDGE

**Значение:** определяет передний фронт булевского выражения

**Синтаксис:** <edge> := REDGE ( <boo\_expression>, <memo\_variable> );

**Операнды:** Первый операнд - любая булевская переменная или сложное выражение, второй операнд - внутренняя переменная, используемая для хранения последнего состояния выражения

**Возвращаемое значение:** TRUE если значение выражения меняется с FALSE на TRUE.  
FALSE во всех остальных случаях.

Передний фронт не может быть определен при помощи оператора REDGE более одного раза в течении одного цикла. Этот оператор можно использовать для описания состояния присоединенного к SFC переходу.

Предупреждение: Булевская переменная для хранения последнего состояния выражения не может быть использована в качестве триггера для фронтов нескольких выражений.

Если выражение - это булевская переменная с именем "xxx", то нужно объявить уникальное имя внутренней переменной "EDGE\_xxx" и использовать его в выражении **REDGE** для этой переменной. Этот метод гарантирует, что эта переменная не будет изменена во время других вычислений REDGE.

Пример:

(\*ST программа, использующая REDGE оператор\*)

(\*Эта программа считает передние фронты булевского входа\*)

(\*Bi120 - переменная булевского входа\*)

(\*Edge\_Bi120 - память состояние переменной Bi120\*)

```
If REDGE (Bi120, Edge_Bi120) Then  
    Counter := Counter + 1;
```

```
End_if;
```

Замечание: Этот оператор не из IEC1131-3. Вы можете использовать стандартный блок R\_TRIG. Он сохраняется для совместимости.

### ═ **"FEDGE" оператор**

**Имя:** FEDGE

**Значение:** определяет задний фронт булевского выражения

**Синтаксис:** <edge> := FEDGE ( <boo\_expression>, <memo\_variable> );

**Возвращаемое значение:** TRUE если значение выражения меняется с TRUE на FALSE.  
FALSE во всех остальных случаях.

Задний фронт не может быть определен при помощи оператора FEDGE более одного раза в течении одного цикла. Этот оператор можно использовать для описания состояния присоединенного к SFC переходу.

Предупреждение: Булевская переменная для хранения последнего состояния выражения не может быть использована в качестве триггера для фронтов нескольких выражений.

Если выражение - это булевская переменная с именем "xxx", то нужно объявить уникальное имя внутренней переменной "EDGER\_xxx" и использовать его в выражении **FEDGE** для этой переменной. Этот метод гарантирует, что эта переменная не будет изменена во время других вычислений FEDGE.

Пример:

(\*ST программа, использующая FEDGE оператор\*)

(\*Эта программа считает передние фронты булевского входа\*)

(\*Bi120 - переменная булевского входа\*)

(\*Edge\_Bi120 - память состояние переменной Bi120\*)

```
If FEDGE (Bi120, Edge_Bi120) Then
    Counter := Counter + 1;
End_if;
```

Замечание: Этот оператор не из IEC1131-3. Вы можете использовать стандартный блок F\_TRIG. Он сохраняется для совместимости.

## В.7.5 Основные операторы ST

Основные операторы языка ST:

Присвоение

Оператор RETURN

Структура IF-THEN-ELSIF-ELSE

Оператор CASE

Итерационный оператор WHILE

Итерационный оператор REPEAT

Итерационный оператор FOR

Оператор EXIT

### ⇒ **Присвоение**

**Name:** :=

**Значение:** присваивает значение выражению

**Синтаксис:** <variable> := <any\_expression>;

**Операнды:** переменная должна быть внутренняя или выход, переменная и выражение должны иметь один и тот же тип

Выражение может быть вызовом подпрограммы или функции из библиотеки ISaGRAF.

Пример:

(\* ST программа с присвоением \*)

(\* variable <=< variable \*)

bo23 := bo10;

(\* variable <=< expression \*)

bo56 := bx34 OR alm100 & (level >= over\_value);

result := (100 \* input\_value) / scale;

(\* присвоение возвращаемого значения подпрограммы \*)

rc := PSelect ( );

(\* присвоение с вызовом функции \*)

limited\_value := min (16, max (0, input\_value) );

## **⇨      Оператор RETURN**

**Имя:**                      **RETURN**

**Значение:**                заканчивает выполнение текущей программы

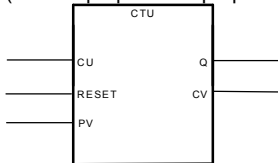
**Синтаксис:**              **RETURN;**

**Операнды:**                (нет)

В блоке SFC оператор RETURN определяет конец выполнения только данного блока.

Пример:

(\* FBD программа: программируемый счетчик \*)



(\* ST программа \*)

If not (CU) then

    Q := false;

    CV := 0;

    RETURN; (\* закончить программу\*)

end\_if;

if R then

    CV := 0;

else

    if (CV < PV) then

        CV := CV + 1;

    end\_if;

end\_if;

Q := (CV >= PV);

## **⇨      Структура IF-THEN-ELSIF-ELSE**

**Имя:**                      **IF... THEN... ELSIF... THEN... ELSE... END\_IF**

**Значение:** выполняет один или два списка ST операторов, выбор осуществляется в соответствии со значением булевского выражения

**Синтаксис:**

```
IF <boolean_expression> THEN
  <statement> ;
  <statement> ;
  ...
ELSIF <boolean_expression> THEN
  <statement> ;
  <statement> ;
  ...
ELSE
  <statement> ;
  <statement> ;
  ...
END_IF;
```

Операторы ELSE и ELSIF - дополнительные. Если ELSE опущен и условие равно FALSE, то никаких инструкций не выполняется.

Пример:

(\*ST программа , использующая оператор IF\*)

```
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 структура без ELSE \*)

```
If overflow THEN
  alarm_level := true;
END_IF;
```

## **⇨ Оператор CASE**

**Имя:** CASE... OF... ELSE... END\_CASE

**Значение:** выполняет один или несколько списков ST операторов, выбор осуществляется в соответствии с целым выражением

**Синтаксис:**

```
CASE <integer_expression> OF
  <value> : <statements> ;
  <value> , <value> : <statements> ;
  ...
ELSE
  <statements> ;
END_CASE;
```

Значением CASE должна быть целая константа. Несколько значений разделенных запятыми могут указывать на один и тот же список операторов. Оператор ELSE - дополнительный.

Пример:

(\*ST программа , использующая оператор CASE\*)

```
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;
```

### == **Итерационный оператор WHILE**

**Имя:** WHILE... DO... END\_WHILE

**Значение:** итерационная структура для группы ST операторов, условие вычисляется прежде выполнения итерации

**Синтаксис:** WHILE <boolean\_expression> DO  
<statement> ;  
<statement> ;  
...  
END\_WHILE ;

Предупреждение: Так как ISaGRAF **синхронная** система входные переменные не обновляются во время итераций WHILE. Изменение состояния входных переменных не может быть использовано для описания условия оператора WHILE.

Пример:

(\*ST программа , использующая оператор WHILE\*)  
(\* эта программа использует "C" функции для чтения \*)  
(\* символов из последовательного порта \*)

```
string := ""; (* empty string *)
nbchar := 0;

WHILE ((nbchar < 16) & ComIsReady ( )) DO
    string := string + ComGetChar ( );
    nbchar := nbchar + 1;
END_WHILE;
```

### == **Итерационный оператор REPEAT**

**Имя:** REPEAT... UNTIL... END\_REPEAT

**Значение:** итерационная структура для группы ST операторов, условие вычисляется после выполнения итерации

**Синтаксис:**

```

REPEAT
  <statement> ;
  <statement> ;
  ...
UNTIL <boolean_condition>
END_REPEAT ;

```

Предупреждение: Так как ISaGRAF **синхронная** система входные переменные не обновляются во время итераций REPEAT. Изменение состояния входных переменных не может быть использовано для описания условия оператора REPEAT.

Пример:

(\*ST программа , использующая оператор REPEAT\*)  
 (\* эта программа использует "C" функции для чтения \*)  
 (\* символов из последовательного порта \*)

```

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;

```

## == Оператор FOR

**Имя:** FOR... TO... BY... DO... END\_FOR  
**Значение:** выполняет ограниченное число итераций, используя целую аналоговую индексную переменную

**Синтаксис:**

```

FOR <index> := <mini> TO <maxi> BY <step> DO
  <statement> ;
  <statement> ;
END_FOR;

```

**Операнды:**

- index:** внутренняя аналоговая переменная, увеличивающаяся на каждом витке
- mini:** начальное значение для индекса (перед первой итерацией)
- maxi:** максимально-допустимое значение индекса
- step:** приращение индекса на каждом шаге

Оператор [BY step] - дополнительный. Если он не определен то приращение равно 1.  
 "While" эквивалент оператора FOR:

```

index := mini;
while (index <= maxi) do
  <statement> ;
  <statement> ;

```

```
index := index + step;  
end_while;
```

Пример:

(\*ST программа , использующая оператор FOR\*)  
(\* эта программа извлекает целые числа из строки \*)

```
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;
```

## ⇒ **Оператор EXIT**

**Имя:** EXIT  
**Значение:** выход из итерационных операторов FOR, WHILE, REPEAT  
**Синтаксис:** EXIT;  
**Операнды:** (нет)

EXIT, обычно, используется в операторе IF внутри блоков FOR, WHILE, REPEAT.

Пример:

(\*ST программа , использующая оператор EXIT\*)  
(\* эта программа ищет символ в строке \*)

```
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;
```

## **В.7.6 Расширения ST**

Следующие функции являются расширениями языка ST:

TSTART-TSTOP :управление таймером

Следующие операторы и функции предназначены для управления исполнением программ наследников SFC:

GSTART	запускает SFC программу
GKILL	убивает SFC программу
GFREEZE	замораживает SFC программу
GRST	перезапускает замороженную SFC программу
GSTATUS	получает текущее состояние SFC программы

Предупреждение: эти функции не относятся к стандарту IEC 1131-3.



Простой эквивалент для GSTART и GKILL может быть получен путем использования синтаксиса SFC шагов:

```
child_name(S); (* эквивалентно GSTART(child_name); *)
child_name(R); (* эквивалентно GKILL(child_name); *)
```

Следующие поля могут быть использованы для доступа к состоянию SFC шага:

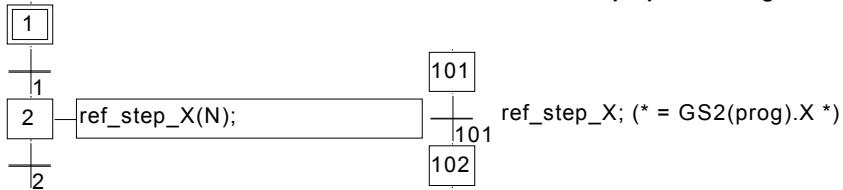
**GSnnn.x** булевская переменная, представляющая активность шага  
**GSnnn.t** время прошедшее с момента последней активизации шага  
 ("nnn" номер SFC шага)

Есть возможность, также проверить активность шага объявленного в другой SFC программе, используя следующий синтаксис:

### GSnnn(progname).x

Предупреждение: номер шага другой программы, при использовании такого синтаксиса не относится к стандарту IEC 1131-3. Простой способ сделать тоже самое, используя правила IEC - это объявить глобальную булевскую переменную в словаре, которая будет представлять активность интересующего нас шага (например ref\_step\_X). Затем вы вводите в шаг переменную с признаком N (ref\_step\_X(N)). Затем в программе, которая желает проверить активность этого шага, вы используете эту переменную.

**Программа Prog** **Другая программа, которой нужна активность шага программы Prog**



### **TSTART оператор**

**Имя:** TSTART

**Значение:** запускает переменную таймер, таймер не модифицируется командой TSTART, т. е. счет начинается с текущего значения таймера

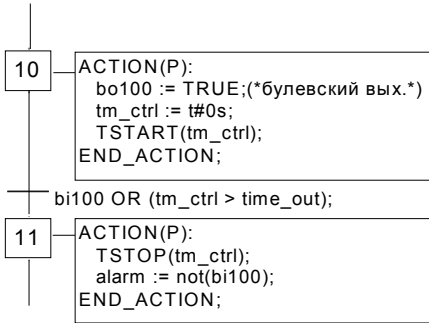
**Синтаксис:** TSTART (<time\_variable> );

**Операнды:** Любой неактивный таймер

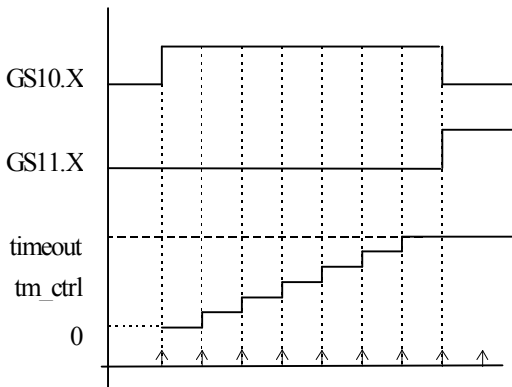
**Возвращаемое значение:** (нет)

Пример:

(\*ST программа , использующая оператор TSTART и TSTOP\*)



Временная диаграмма если bi100 всегда равно FALSE:



Таймер сохраняет свое значение в течении одного цикла.

### ≡ **TSTOP оператор**

**Имя:** TSTOP

**Значение:** останавливает переменную таймер, таймер не модифицируется командой TSTOP.

**Синтаксис:** TSTOP (<time\_variable> );

**Операнды:** Любой активный таймер

**Возвращаемое значение:** (нет)

Пример: см. TSTART

### ≡ **GSTART оператор**

**Имя:** GSTART

**Значение:** запускает программу наследник, устанавливая маркер на все ее начальные шаги.

**Синтаксис:** GSTART (<child\_program> );

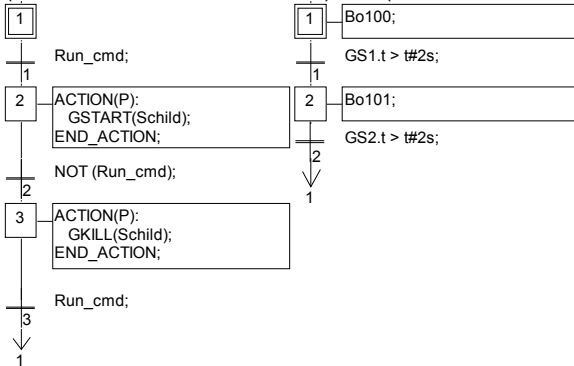
**Операнды:** SFC программа, которая должна быть наследником по отношению к программе в которой написан оператор

**Возвращаемое значение:** (нет)

Наследники программы-наследника не запускаются автоматически оператором GSTART.  
Замечание: Так как оператор GSTART не относится к стандарту IEC 1131-3, для запуска программы-наследника лучше использовать признак S:  
Child\_name(S);

Пример использование GSTART и GKILL:

(\* Последовательность 'Sfather' \*) (\* Последовательность 'Schild' \*)



## **GKILL оператор**

**Имя:** GKILL

**Значение:** убивает SFC программу, уничтожая маркер на всех ее текущих шагах.

**Синтаксис:** GKILL (<child\_program> );

**Операнды:** SFC программа, которая должна быть наследником по отношению к программе, в которой написан оператор

**Возвращаемое значение:** (нет)

Наследники программы-наследника убиваются автоматически оператором GKILL.  
Замечание: Так как оператор GKILL не относится к стандарту IEC 1131-3, для того чтобы убить программу-наследник лучше использовать признак R:  
Child\_name(R);

Пример: см. GSTART (функция описана выше)

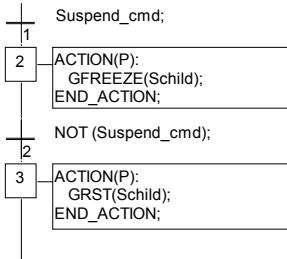
## **GFREEZE оператор**

**Имя:** GFREEZE

**Значение:** удаляет все существующие маркеры программы-наследника и запоминает их положение так, что программа может быть перезапущена с помощью оператора GRST.

**Синтаксис:** **GFREEZE (<child\_program> );**  
**Операнды:** SFC программа, которая должна быть наследником по отношению к программе в которой написан оператор

**Возвращаемое значение:** (нет)  
 Наследники программы-наследника замораживаются автоматически вместе с указанной программой.  
 Замечание: Оператор GFREEZE не относится к стандарту IEC 1131-3.  
 Пример:



### **GRST оператор**

**Имя:** **GRST**  
**Значение:** перезапускает программу-наследник и замороженную оператором GFREEZE, все маркеры, уничтоженные оператором GFREEZE восстанавливаются.  
**Синтаксис:** **GRST (<child\_program> );**  
**Операнды:** SFC программа, которая должна быть наследником по отношению к программе в которой написан оператор

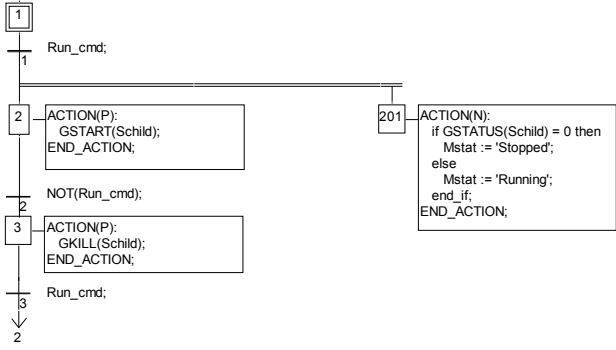
**Возвращаемое значение:** (нет)  
 Наследники программы-наследника запускаются автоматически вместе с указанной программой.  
 Замечание: Оператор GRST не относится к стандарту IEC 1131-3.  
 Пример: см. GFREEZE (функция описана выше)

### **GSTATUS оператор**

**Имя:** **GSTATUS**  
**Значение:** возвращает текущее состояние SFC программы.  
**Синтаксис:** **<ana\_var> := GSTATUS (<child\_program> );**  
**Операнды:** SFC программа, которая должна быть наследником по отношению к программе в которой написан оператор

**Возвращаемое значение:** 0 = программа неактивна (убита)  
 1 = программа активна (запущена)  
 2 = программа заморожена

Замечание: Оператор GSTATUS не относится к стандарту IEC 1131-3.  
 Пример:



## В.8 Язык инструкций (IL)

Список инструкций или IL - это язык низкого уровня. Инструкции всегда относятся к **текущему результату** (или IL регистру). Оператор определяет операцию, которая должна быть выполнена с текущим результатом и операндом. Результат операции снова запоминается в текущем результате.

### В.8.1 Основной синтаксис IL

IL программа - это список **инструкций**. Каждая инструкция должна начинаться с новой строки и должна содержать **оператор**, с дополнительным модификаторами, если нужно, для специфических операций, один или несколько операндов, разделенных запятой (','). Инструкции может предшествовать **метка** с двоеточием (':'). Если к инструкции присоединен комментарий, то он должен находиться в конце строки. Комментарий всегда начинается с ('\*' и заканчивается '\*'). Между инструкциями может быть введена пустая строка. Комментарии могут быть помещены в пустые строки.

Метка	Оператор	Операнд	Комментарий
Start:	LD	IX1	(* нажать кнопку *)
	ANDN	MX5	(* команда разрешена*)
	ST	QX2	(* запустить мотор *)

#### ☰ **Метки**

Инструкции может предшествовать **метка** с двоеточием (':'). Метка может быть помещена на пустую строку. Метки используются в качестве операндов для некоторых операций, таких как прыжки. Имена меток должны удовлетворять следующим правилам:

- имя не может быть длиннее **16** символов
- первым символом должна быть **буква**
- последующими символами могут быть **буквы, цифры** или **символ подчеркивания**

В одной программе одно и то же имя не может быть использовано для обозначения более чем одной метки. Имя метки может совпадать с именем переменной.

#### ☰ **Модификаторы оператора**

Ниже представлены возможные модификаторы оператора. Символ модификатора должен завершать имя оператора, без пробелов между ними.

<b>N</b>	булевское отрицание операнда
<b>(</b>	задержанная операция
<b>C</b>	условная операция

Модификатор 'N' определяет булевское отрицание операнда Например, инструкция **ORN IX12** интерпретируется как: **result := result OR NOT (IX12)**.

Модификатор скобка '(' указывает на то, что выполнение инструкции должно быть задержано до тех пор пока не встретится закрытая скобка ')

Модификатор "C" указывает на то, что соответствующая инструкция должна быть выполнена только если текущий результат имеет значение TRUE (отличное от нуля).

Модификатор 'C', может комбинироваться с модификатором 'N', который указывает на то, что инструкция должна быть выполнена только если текущий результат имеет значение FALSE (0).

## ⇒ **Задержанные операции**

Из-за того, что существует только один IL регистр, некоторые операции могут быть задержаны, так что порядок исполнения инструкций может быть изменен. Скобки используются для того, чтобы выделить задержанные операции:

'(' модификатор указывает операцию, которая должна быть задержана  
)' оператор выполняет задержанную операцию

Открывающаяся скобка '(' указывает на то, что выполнение инструкции должно быть задержано до тех пор пока не встретится закрытая скобка ')'. Например, следующая последовательность:

```

AND( IX12
OR IX35
)

```

интерпретируется как:

**result := result AND ( IX12 OR IX35 )**

## **В.8.2 Операторы IL**

Следующая таблица представляет стандартные операторы языка IL:

Оператор	Модификатор	Операнд	Описание
LD	N	переменная, константа	Загружает операнд
ST	N	переменная	Запоминает текущий результат
S		BOO переменная	Устанавливает на TRUE
R		BOO переменная	Сбрасывает на FALSE
AND	N (	BOO	логическое И
&	N (	BOO	логическое И
OR	N (	BOO	логическое ИЛИ
XOR	N (	BOO	исключающее ИЛИ
ADD	(	переменная, константа	Сложение
SUB	(	переменная, константа	Вычитание
MUL	(	переменная, константа	Умножение
DIV	(	переменная, константа	Деление

GT	(	переменная, константа	Проверить: >
GE	(	переменная, константа	Проверить: >=
EQ	(	переменная, константа	Проверить: =
LE	(	переменная, константа	Проверить: <=
LT	(	переменная, константа	Проверить: <
NE	(	переменная, константа	Проверить: <>
CAL	C N	Экземпляр функционального блока	Вызов функционального блока
JMP	C N	имя метка	Прыжок на метку
RET	C N		Возврат из подпрограммы
)			Выполнить задержанную операцию

В следующей главе описаны операторы, которые являются специфическими для языка IL, другие стандартные операторы можно найти в главе “Стандартные операторы, функциональные блоки и функции”.

### Оператор LD

**Операция:** загружает значение в текущий результат

**Допустимые модификаторы:** N

**Операнд:** константа, внутренняя, входная или выходная переменная

Пример:

(\* ПРИМЕР LD ОПЕРАЦИЙ \*)

```
LDex:    LD      false      (* результат := FALSE boolean constant *)
         LD      true       (* результат := TRUE boolean constant *)
         LD      123        (* результат := integer constant *)
         LD      123.1      (* результат := real constant *)
         LD      t#3ms      (* результат := time constant *)
         LD      boo_var1   (* результат := boolean variable *)
         LD      ana_var1   (* результат := analog variable *)
         LD      tmr_var1   (* результат := timer variable *)
         LDN     boo_var2   (* результат := NOT ( boolean variable ) *)
```

### Оператор ST

**Операция:** запоминает текущий результат в переменной

Текущий результат этим оператором не изменяется

**Допустимые модификаторы:** N

**Операнд:** внутренняя или выходная переменная

Пример:

(\* ПРИМЕР ST ОПЕРАЦИЙ \*)

```
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 \*)

## Оператор S

**Операция:** запоминает TRUE в булевой переменной, если текущий результат имеет значение TRUE. Никаких операций не выполняется, если текущий результат равен FALSE. Текущий результат не модифицируется.

**Допустимые модификаторы:** (нет)

**Операнд:** внутренняя или выходная булевская переменная

Пример:

```
(* ПРИМЕР S ОПЕРАЦИЙ *)
SETex:   LD   true      (*текущий результат := TRUE *)
         S    boo_var1  (* boo_var1 := TRUE *)
         LD   false     (*текущий результат не изменяется *)
         S    boo_var1  (*текущий результат := FALSE *)
         S    boo_var1  (*ничего не делать - boo_var1 не изменяется *)
```

## Оператор R

**Операция:** запоминает FALSE в булевой переменной если текущий результат имеет значение TRUE. Никаких операций не выполняется если текущий результат равен FALSE. Текущий результат не модифицируется.

**Допустимые модификаторы:** (нет)

**Операнд:** внутренняя или выходная булевская переменная

Пример:

```
(* ПРИМЕР R ОПЕРАЦИЙ *)
RESETex: LD   true      (*текущий результат:= TRUE *)
         R    boo_var1  (* boo_var1 := FALSE *)
         LD   true      (*текущий результат не изменяется *)
         ST   boo_var2  (* boo_var2 := TRUE *)
         LD   false     (*текущий результат:= FALSE *)
         R    boo_var1  (*ничего не делать - boo_var1 не изменяется *)
```

## Оператор JMP

**Операция:** прыгает на указанную метку

**Допустимые модификаторы:** C N

**Операнд:** метка определенная в той же IL программе

Пример:

(\* следующий пример проверяет значение аналогового селектора (0 или 1 или 2)  
(\* чтобы установить один из 3 булевских выходов. Проверка "равно 0" делается  
(\* оператором JMP C \*)

```
JMPex:   LD   selector  (* selector is 0 or 1 or 2 *)
         BOO          (* превращение в boolean *)
         JMP C test1    (* if selector = 0 then *)
         LD   true
         ST   bo0       (* bo0 := true *)
         JMP JMPend    (* конец программы *)
```

```

test1:      LD      selector
            SUB      1      (* уменьшить selector: теперь 0 или 1 *)
            BOO      (* превращение в boolean *)
            JMP     test2  (* if selector = 0 then *)
            LD      true
            ST      bo1   (* bo1 := true *)
            JMP     JMPend (* конец программы*)
test2:      LD      true  (* последняя возможность *)
            ST      bo2   (* bo2 := true *)
            JMP     end:  (* конец IL программы*)

```

## ▬ Оператор RET

**Операция:** заканчивает текущий список инструкций. Если IL последовательность - подпрограмма, то текущий результат возвращается в вызывающую подпрограмму.

**Допустимые модификаторы:** C N

**Операнд:** (нет)

Пример:

(\* следующий пример проверяет значение аналогового селектора (0 или 1 или 2)  
 (\* чтобы установить один из 3 булевских выходов. Проверка "равно 0" делается  
 (\* the JMPc operator

```

JMPex:      LD      selector  (* selector равен 0 или 1 или 2 *)
            BOO      (* превращение в boolean *)
            JMP     test1  (* if selector = 0 then *)
            LD      true
            ST      bo0   (* bo0 := true *)
            RET      (* end - return 0 *)
            (* уменьшить selector *)
test1:      LD      selector
            SUB      1      (* selector теперь 0 или 1 *)
            BOO      (* превращение в boolean *)
            JMP     test2  (* if selector = 0 then *)
            LD      true
            ST      bo1   (* bo1 := true *)
            LD      1      (* загрузить действительное значение selector *)
            RET      (* end - return 1 *)
            (* последняя возможность *)
test2:      RETNC      (* возвращает если selector содержит *)
            (* неправильное значение *)
            LD      true
            ST      bo2   (* bo2 := true *)
            LD      2      (* загрузить действительное значение selector *)
            (* конец - возврат 2 *)

```

## ▬ Оператор “)”

**Операция:** выполняет задержанную операцию. Задержанная операция обозначается “(“

**Допустимые модификаторы:** (нет)

**Операнд:** (нет)

Пример:

(\* Следующая программа использует задержанные операции: \*)

(\* res := a1 + (a2 \* (a3 - a4) \* a5) + a6; \*)

Delayed:	LD	a1	(* result := a1; *)
	ADD(	a2	(* задержанный ADD - result := a2; *)
	MUL(	a3	(* задержанный MUL - result := a3; *)
	SUB	a4	(* result := a3 - a4; *)
	)		(*выполнить задержан. MUL-result:=a2*(a3-a4); *)
	MUL	a5	(* result := a2 * (a3 - a4) * a5; *)
	)		(* выполнить задержанный ADD *)
			(* result := a1 + (a2 * (a3 - a4) * a5); *)
	ADD	a6	(* result := a1 + (a2 * (a3 - a4) * a5) + a6; *)
	ST	res	(*запомнить текущ. результат в переменной res *)

## ≡ **Вызов подпрограмм или функций**

Подпрограмма или функция (написанная на любом из языков IL, ST, LD, FBD или "C") вызывается из языка IL, используя имя в качестве оператора.

**Операция:** выполняет подпрограмму или функцию - значение возвращенное подпрограммой или функцией запоминается в текущем результате IL

**Допустимые модификаторы:** (нет)

**Операнд:** Первый параметр должен быть запомнен в текущем результате перед вызов. Остальные параметры записываются в поле операнда через запятую

Пример:

(\* Вызывающая программа: превращает аналоговое значение в временное \*)

Main:	LD	bi0	
	SUBPRO	bi1,bi2	(*вызвать подпрограмму чтобы получить аналоговое значение *)
	ST	result	(* result := значение возвращенное подпрограммой *)
	GT	vmax	(*проверка переполнения *)
	RETC		(*возврат по переполнению *)
	LD	result	
	MUL	1000	(*превратить секунды в миллисекунды *)
	TMR		(* превратить в таймер *)
	ST	tmval	(* запоминает превращенное значение в таймере*)

(\* Вызванная подпрограмма 'SUBPRO' : вычисляет аналоговое значение \*)

(\* заданное как двоичное число трех булевских входов: in0, in1, in2 - три параметра подпрограммы\*)

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 (* вернуть текущий результат в вызывающую программу*)

```

## ≡ **Вызов функциональных блоков: оператор CAL**

**Операция:** вызывает функциональный блок

**Допустимые модификаторы:** C N

**Операнд:** Имя функционального блока. Входные параметры блока должны быть присвоены с помощью операторов LD/ST до вызова. Выходные параметры могут быть использованы.

Пример 1:

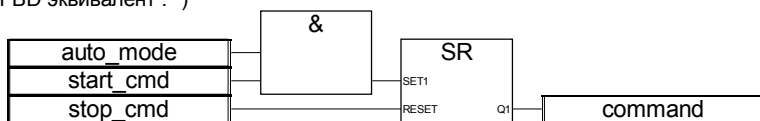
(\* Вызов функционального блока SR : SR1 - экземпляр SR \*)

```

LD      auto_mode
AND     start_cmd
ST      SR1.set1
LD      stop_cmd
ST      SR1.reset
CAL     SR1
LD      SR1.Q1
ST      command

```

(\* FBD эквивалент : \*)



Пример 2

(\*Мы предполагаем, что R\_TRIG1 - это экземпляр блока R\_TRIG и CTU1 - это экземпляр блока CTU\*)

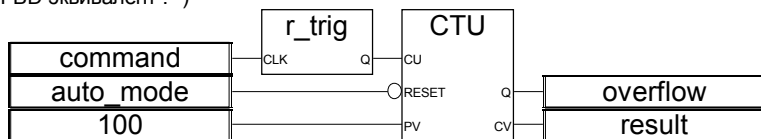
```

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 эквивалент : \*)



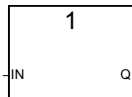
## В.9 Стандартные операторы, функциональные блоки и функции

### В.9.1 Стандартные операторы

Существуют следующие стандартные операторы IEC языков:

Работа с данными .....	Присвоение, Аналоговое отрицание
Булевские операции .....	Логическое И (AND) Логическое ИЛИ (OR) Логическое исключающее ИЛИ (OR)
Арифметические операции.....	Сложение Вычитание Умножение Деление
Логические операции.....	Аналоговое побитовое И (AND) Аналоговое побитовое ИЛИ (OR) Аналоговое побитовое исключающее ИЛИ (OR) Аналоговое побитовое отрицание
Сравнения .....	Меньше чем Меньше или равно Больше чем Больше или равно Равно Неравно
Превращение данных.....	Превратить в Boolean Превратить в Integer Analog Превратить в Real Analog Превратить в Timer Превратить в Message
Другие .....	Соединение строк Доступ к системе Работа с каналами В/В

#### 1 gain



Аргументы:

**IN**

любой тип

**Q**

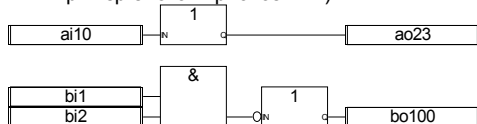
любой тип

Описание:

Приписывает значение одной переменной другой переменной.

Этот блок очень удобен для прямой связи диаграммы входа с диаграммой выхода. Он может быть также использован (с линией булевского отрицания) для инвертирования состояния линии соединенной с диаграммой выхода.

(\*FBD пример блоков присвоения\*)



(\* ST эквивалент: \*)

ao23 := ai10;

bo100 := NOT (bi1 AND bi2);

(\* IL эквивалент: \*)

LD ai10

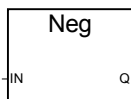
ST ao23

LD bi1

AND bi2

STN bo100

## NEG



Аргументы:

**IN**

INT-REAL вход и выход должны иметь один и тот же формат

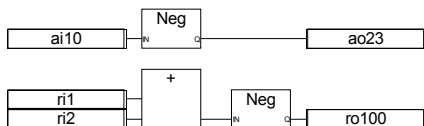
**Q**

INT-REAL

Описание:

Приписывает отрицание переменной.

(\*FBD пример блоков отрицания\*)



(\* ST эквивалент: \*)

ao23 := - (ai10);

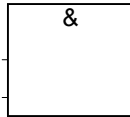
ro100 := - (ri1 + ri2);

```

(* IL эквивалент: *)
LD      ai10
MUL     -1
ST      ao23
LD      r1
ADD     r2
MUL     -1.0
ST      ro100

```

## & AND



Замечание: Для этого оператора количество входов может быть больше чем два.

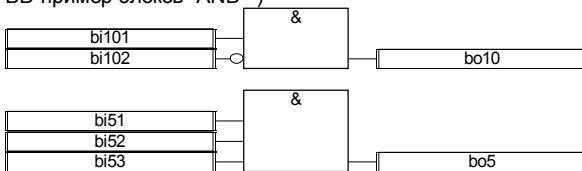
Аргументы:

(inputs)	BOOLEAN	
output	BOOLEAN	логическое И двух или более входов

Описание:

Логическое И двух или больше значений.

(\*FBD пример блоков "AND"\*)



(\* ST эквивалент: \*)

```

bo10 := bi101 AND NOT (bi102);
bo5 := (bi51 AND bi52) AND bi53;

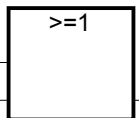
```

(\* IL эквивалент \*)

LD	bi101	(* текущий результат := bi101 *)
ANDN	bi102	(*текущий результат:= bi101 AND not(bi102) *)
ST	bo10	(* bo10 := текущий результат *)
LD	bi51	(*текущий результат:= bi51;
&	bi52	(*текущий результат:= bi51 AND bi52 *)
&	bi53	(* тек. результат := (bi51 AND bi52) AND bi53 *)
ST	bo5	(* bo5 := текущий результат *)

## >=1 OR





Замечание: Для этого оператора количество входов может быть больше чем два.

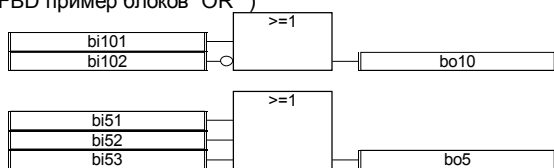
Аргументы:

(inputs)	BOOLEAN	
output	BOOLEAN	логическое ИЛИ двух или более входов

Описание:

Логическое ИЛИ двух или больше значений.

(\*FBD пример блоков "OR"\*)



(\* ST эквивалент: \*)

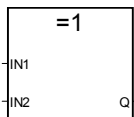
bo10 := bi101 OR NOT (bi102);

bo5 := (bi51 OR bi52) OR bi53;

(\* IL эквивалент: \*)

```
LD      bi101
ORN     bi102
ST      bo10
LD      bi51
OR      bi52
OR      bi53
ST      bo5
```

## =1 XOR



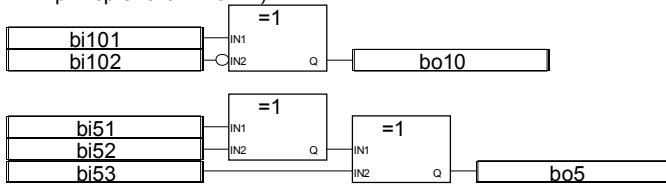
Аргументы:

<b>IN1</b>	BOOLEAN	
<b>IN2</b>	BOOLEAN	
<b>Q</b>	BOOLEAN	логическое ИЛИ двух или более входов

Описание:

Логическое ИЛИ двух или больше значений.

(\*FBD пример блоков "XOR"\*)



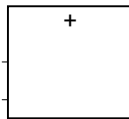
(\* ST эквивалент: \*)

bo10 := bi101 XOR NOT (bi102);  
bo5 := (bi51 XOR bi52) XOR bi53;

(\* IL эквивалент: \*)

```
LD      bi101
XORN   bi102
ST      bo10
LD      bi51
XOR    bi52
XOR    bi53
ST      bo5
```

+



Замечание: Для этого оператора количество входов может быть больше чем два.

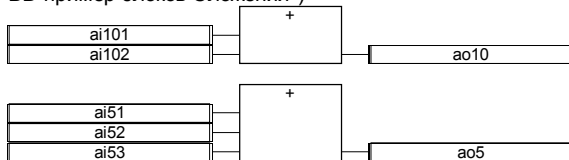
Аргументы:

(inputs)	INT-REAL	могут быть целыми или действительными (все входы должны быть одного формата)
output	INT-REAL	знаковое сложение входов

Описание:

Складывает две или несколько переменных.

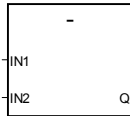
(\*FBD пример блоков Сложения\*)



(\* ST эквивалент: \*)

ao10 := ai101 + ai102;  
ao5 := (ai51 + ai52) + ai53;

```
(* IL эквивалент: *)
LD      ai101
ADD     ai102
ST      ao10
LD      ai51
ADD     ai52
ADD     ai53
ST      ao5
```



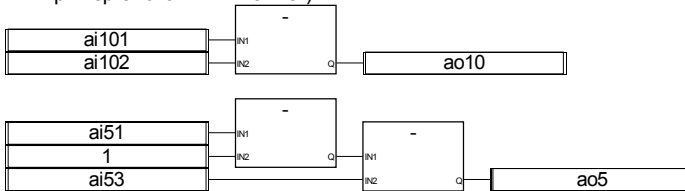
Аргументы:

<b>IN1</b>	INT-REAL	могут быть целыми или действительными
<b>IN2</b>	INT-REAL	(IV1 и IN2 должны быть одного формата)
<b>Q</b>	INT-REAL	вычитание (первый - второй)

Описание:

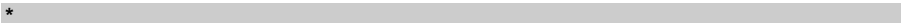
Вычитает две переменные (первый - второй).

(\*FBD пример блоков Вычитание\*)

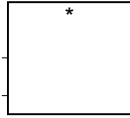


```
(* ST эквивалент: *)
ao10 := ai101 - ai102;
ao5 := (ai51 - 1) - ai53;
```

```
(* IL эквивалент: *)
LD      ai101
SUB     ai102
ST      ao10
LD      ai51
SUB     1
SUB     ai53
ST      ao5
```



\*



Замечание: Для этого оператора количество входов может быть больше чем два.

Аргументы:

(inputs)

INT-REAL

могут быть целыми или действительными  
(все входы должны быть одного формата)

output

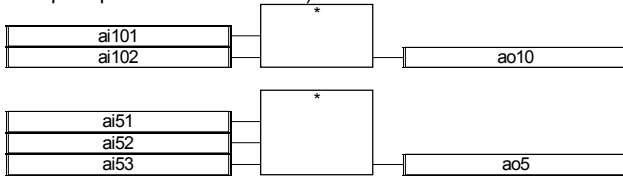
INT-REAL

знаковое сложение входов

Описание:

Умножает две или несколько переменных.

(\*FBD пример блоков Умножение\*)



(\* ST эквивалент \*)

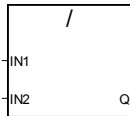
ao10 := ai101 \* ai102;

ao5 := (ai51 \* ai52) \* ai53;

(\* IL эквивалент: \*)

```
LD    ai101
MUL   ai102
ST    ao10
LD    ai51
MUL   ai52
MUL   ai53
ST    ao5
```

/



Аргументы:

**IN1**

INT-REAL

могут быть целыми или действительными

**IN2**

INT-REAL

ненулевое аналоговое значение

(IV1 и IN2 должны быть одного формата)

**Q**

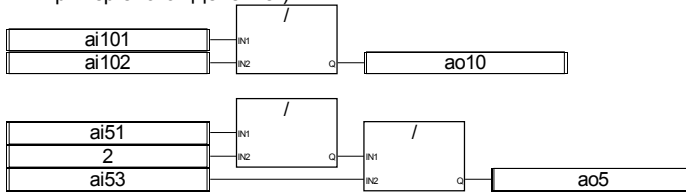
INT-REAL

знаковое целое или действительное деление IN1 на IN2

Описание:

Делит две переменные (первый / второй).

(\*FBD пример блоков Деление\*)



(\* ST эквивалент: \*)

ao10 := ai101 / ai102;

ao5 := (ai5 / 2) / ai53;

(\* IL эквивалент: \*)

LD ai101

DIV ai102

ST ao10

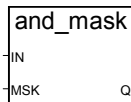
LD ai51

DIV 2

DIV ai53

ST ao5

## AND\_MASK



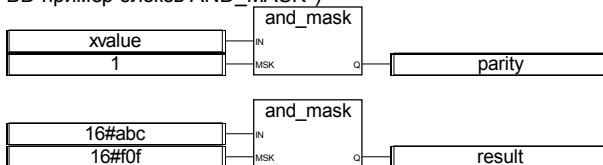
Аргументы:

<b>IN</b>	INT	целый формат
<b>MSK</b>	INT	целый формат
<b>Q</b>	INT	побитовый логический AND между IN и MSK

Описание:

Целый аналоговый побитовый И.

(\*FBD пример блоков AND\_MASK\*)



(\* ST эквивалент: \*)

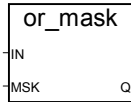
parity := AND\_MASK (xvalue, 1); (\* 1 if xvalue is odd \*)

result := AND\_MASK (16#abc, 16#0f); (\* equals 16#a0c \*)

(\* IL эквивалент: \*)

```
LD      xvalue
AND_MASK 1
ST      parity
LD      16#abc
AND_MASK 16#f0f
ST      result
```

## OR\_MASK



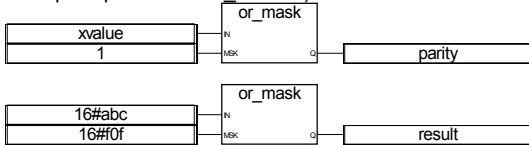
Аргументы:

<b>IN</b>	INT	целый формат
<b>MSK</b>	INT	целый формат
<b>Q</b>	INT	битовый логический OR между IN и MSK

Описание:

Целый аналоговый битовый ИЛИ.

(\*FBD пример блоков OR\_MASK\*)



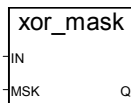
(\* ST эквивалент: \*)

```
is_odd := OR_MASK (xvalue, 1); (* делает значение всегда нечетным *)
result := OR_MASK (16#abc, 16#f0f); (* равно 16#fbf *)
```

(\* IL эквивалент: \*)

```
LD      xvalue
OR_MASK 1
ST      is_odd
LD      16#abc
OR_MASK 16#f0f
ST      result
```

## XOR\_MASK



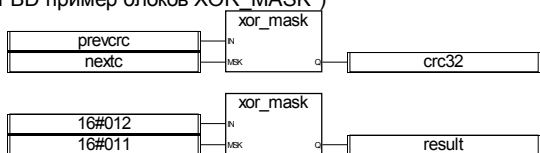
Аргументы:

<b>IN</b>	INT	целый формат
<b>MSK</b>	INT	целый формат
<b>Q</b>	INT	битовый логический исключающий OR между IN и MSK

Описание:

Целый аналоговый побитовый исключающий ИЛИ.

(\*FBD пример блоков XOR\_MASK\*)



(\* ST эквивалент: \*)

crc32 := XOR\_MASK (prevcrc, nextc);

result := XOR\_MASK (16#012, 16#011); (\* equals 16#003 \*)

(\* IL эквивалент: \*)

LD prevcrc

XOR\_MASK nextc

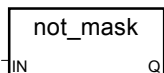
ST crc32

LD 16#012

XOR\_MASK 16#011

ST result

## NOT\_MASK



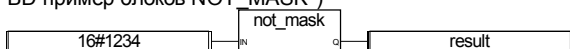
Аргументы:

<b>IN</b>	INT	целый формат
<b>Q</b>	INT	битовое логическое отрицание 32 разрядного IN

Описание:

Целое аналоговое побитовое логическое отрицание.

(\*FBD пример блоков NOT\_MASK\*)



(\*ST эквивалент: \*)

result := NOT\_MASK (16#1234);

(\* result is 16#FFFF\_EDCB \*)

(\* IL эквивалент: \*)

LD 16#1234

NOT\_MASK

ST result

<



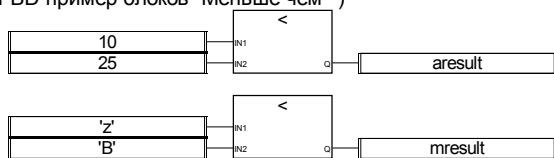
Аргументы:

**IN1** INT-REAL-TMR-MSG  
**IN2** INT-REAL-TMR-MSG вход и выход должны иметь один и тот же тип  
**Q** BOOLEAN TRUE if IN1 < IN2

Описание:

Проверить, что одна величина меньше другой.

(\*FBD пример блоков "Меньше чем"\*)



(\* STЭквивалент: \*)

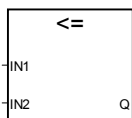
areult := (10 < 25); (\* areult is TRUE \*)

mresult := ('z' < 'B'); (\* mresult is FALSE \*)

(\* IL эквивалент: \*)

LD 10  
 LT 25  
 ST areult  
 LD 'z'  
 LT 'B'  
 ST mresult

<=



Аргументы:

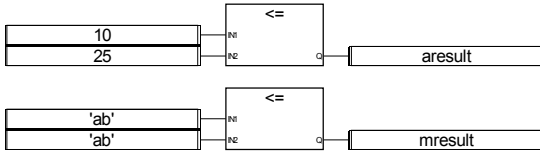
**IN1** INT-REAL-MSG  
**IN2** INT-REAL-MSG вход и выход должны иметь один и тот же тип  
**Q** BOOLEAN TRUE if IN1 <= IN2

Описание:

Проверить, что одна величина меньше или равна другой.

(\*FBD пример блоков "Меньше или равно"\*)

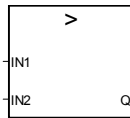




(\* ST эквивалент: \*)  
 areult := (10 <= 25); (\* areult is TRUE \*)  
 mresult := ('ab' <= 'ab'); (\* mresult is TRUE \*)

(\* IL эквивалент: \*)  
 LD 10  
 LE 25  
 ST areult  
 LD 'ab'  
 LE 'ab'  
 ST mresult

>



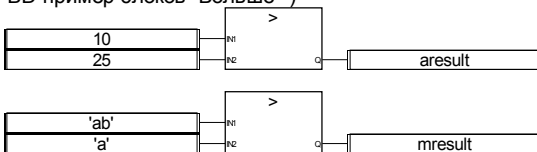
Аргументы:

**IN1** INT-REAL-TMR-MSG  
**IN2** INT-REAL-TMR-MSG вход и выход должны иметь один и тот же тип  
**Q** BOOLEAN TRUE if IN1 > IN2

Описание:

Проверить, что одна величина больше другой.

(\*FBD пример блоков "Больше"\*)

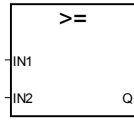


(\* ST эквивалент: \*)  
 areult := (10 > 25); (\* areult is FALSE \*)  
 mresult := ('ab' > 'a'); (\* mresult is TRUE \*)

(\* IL эквивалент: \*)  
 LD 10

GT 25  
 ST aresult  
 LD 'ab'  
 GT 'a'  
 ST mresult

**>=**



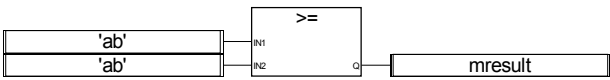
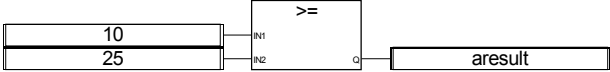
Аргументы:

**IN1** INT-REAL-MSG  
**IN2** INT-REAL-MSG вход и выход должны иметь один и тот же тип  
**Q** BOOLEAN TRUE if IN1 >= IN2

Описание:

Проверить, что одна величина больше или равна другой.

(\*FBD пример блоков "Больше или равно"\*)



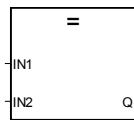
(\* ST эквивалент: \*)

areult := (10 >= 25); (\* areult is FALSE \*)  
 mresult := ('ab' >= 'ab'); (\* mresult is TRUE \*)

(\* IL эквивалент: \*)

LD 10  
 GE 25  
 ST areult  
 LD 'ab'  
 GE 'ab'  
 ST mresult

**=**



Аргументы:

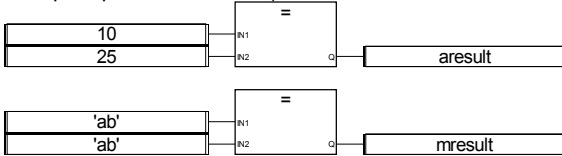
**IN1** INT-REAL-MSG

**IN2** INT-REAL-MSG вход и выход должны иметь один и тот же тип  
**Q** BOOLEAN TRUE if IN1 = IN2

Описание:

Проверить, что одна величина равна другой.

(\*FBD пример блоков "Равно"\*)



(\* ST эквивалент: \*)

areult := (10 = 25); (\* areult is FALSE \*)

mresult := ('ab' = 'ab'); (\* mresult is TRUE \*)

(\* IL эквивалент: \*)

LD 10

EQ 25

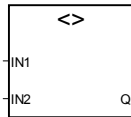
ST areult

LD 'ab'

EQ 'ab'

ST mresult

<>



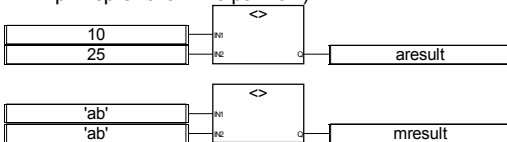
Аргументы:

**IN1** INT-REAL-MSG  
**IN2** INT-REAL-MSG вход и выход должны иметь один и тот же тип  
**Q** BOOLEAN TRUE if IN1 <> IN2

Описание:

Проверить, что одна величина не равна другой.

(\*FBD пример блоков "Не равно"\*)



(\* ST эквивалент: \*)

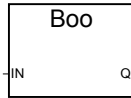
areult := (10 <> 25); (\* areult is TRUE \*)

mresult := ('ab' <> 'ab'); (\* mresult is FALSE \*)

(\* IL эквивалент: \*)

```
LD      10
NE      25
ST      aresult
LD      'ab'
NE      'ab'
ST      mresult
```

## BOO



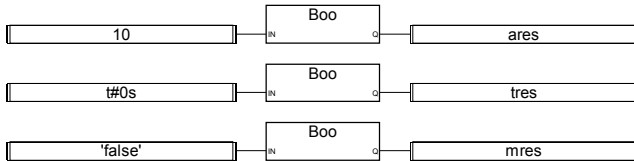
Аргументы:

<b>IN</b>	ANY	любое не булевское значение
<b>Q</b>	BOO	TRUE для ненулевого значения FALSE для нулевого значения TRUE для сообщения 'TRUE' FALSE для сообщения 'FALSE'

Описание:

Превращает переменную в булевую.

(\*FBD пример блоков BOO\*)



(\* ST эквивалент: \*)

```
ares := BOO (10);
tres := BOO (#0s);
mres := BOO ('false');
```

(\* ares равно TRUE \*)

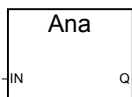
(\* tres равно FALSE \*)

(\* mres равно FALSE \*)

(\* IL эквивалент: \*)

```
LD      10
BOO
ST      ares
LD      #0s
BOO
ST      tres
LD      'false'
BOO
ST      mres
```

## ANA



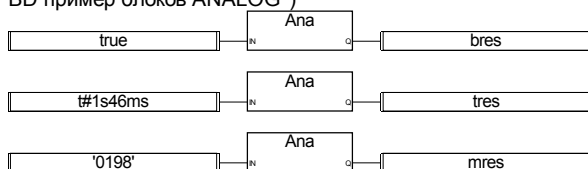
Аргументы:

<b>IN</b>	ANY	любая нецелая аналоговая величина
<b>Q</b>	INT	0 if IN = FALSE / 1 if IN = TRUE
		число миллисекунд для таймера
		целая часть действительного аналогового
		десятичного числа представленное строкой

Описание:

Превращает переменную в целую.

(\*FBD пример блоков ANALOG\*)



(\* ST эквивалент: \*)

bres := ANA (true);

tres := ANA (t#1s46ms);

mres := ANA ('0198');

(\* bres равно 1 \*)

(\* tres равно 1046 \*)

(\* mres равно 198 \*)

(\* IL эквивалент: \*)

LD true

ANA

ST bres

LD t#1s46ms

ANA

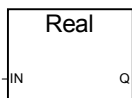
ST tres

LD '0198'

ANA

ST mres

## REAL



Аргументы:

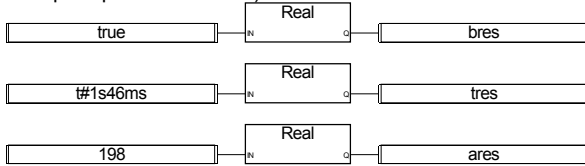
<b>IN</b>	BOO-INT-TMR	любая недействительная аналоговая величина
<b>Q</b>	INT	0.0 if IN = FALSE / 1 if IN = TRUE

число миллисекунд для таймера  
число эквивалентное целому  
аналоговому

Описание:

Превращает переменную в действительную.

(\*FBD пример блоков REAL\*)



(\* ST эквивалент: \*)

bres := REAL (true);

tres := REAL (t#1s46ms);

ares := REAL (198);

(\* bres равно 1.0 \*)

(\* tres равно 1046.0 \*)

(\* ares равно 198.0 \*)

(\* IL эквивалент: \*)

LD true

REAL

ST bres

LD t#1s46ms

REAL

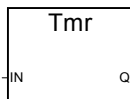
ST tres

LD 198

REAL

ST ares

## TMR



Аргументы:

**IN**

INT-REAL

любая не временная величина

IN (или целая часть от IN, если он действительный) -  
число миллисекунд

**Q**

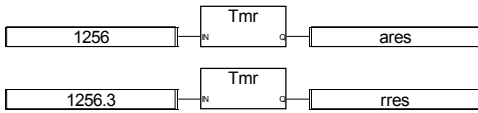
TIMER

временная величина, представленная IN

Описание:

Превращает аналоговую переменную в таймер.

(\*FBD пример блоков TMR\*)



(\* ST эквивалент: \*)

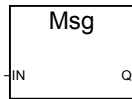
ares := TMR (1256);  
rres := TMR (1256.3);

(\* ares := t#1s256ms \*)  
(\* rres := t#1s256ms \*)

(\* IL эквивалент: \*)

LD 1256  
TMR  
ST ares  
LD 1256.3  
TMR  
ST rres

## MSG



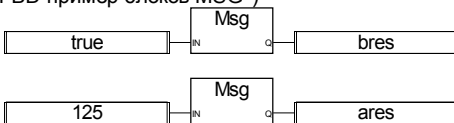
Аргументы:

<b>IN</b>	BOO-INT-REAL	любая не строковая величина
<b>Q</b>	MSG	'false' или 'true' если IN - булевская десятичное представление, если IN - аналоговая

Описание:

Преобразует любую переменную в строку.

(\*FBD пример блоков MSG\*)



(\* ST Эквивалент: \*)

bres := MSG (true); (\* bres равно 'TRUE' \*)  
ares := MSG (125); (\* ares равно '125' \*)

(\* IL эквивалент: \*)

LD true  
MSG  
ST bres  
LD 125  
MSG





Доступ к системным параметрам.

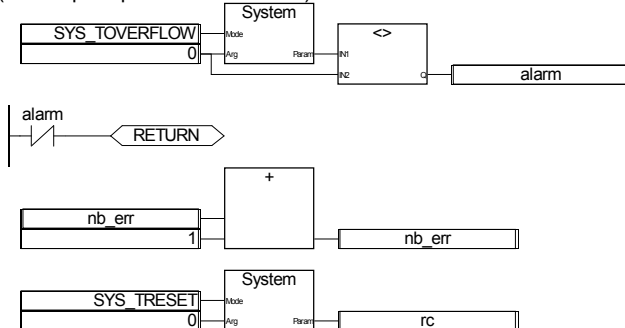
Далее следует список возможных команд для функции SYSTEM:

Команда	Значение
SYS_TALLOWED	читать допустимое время цикла
SYS_TCURRENT	читать текущее время цикла
SYS_TMAXIMUM	читать максимальное время цикла
SYS_TOVERFLOW	читать переполнения времени цикла
SYS_TRESET	сбросить счетчик времени цикла
SYS_TWRITE	изменить время цикла
SYS_ERR_TEST	проверка ошибок выполнения
SYS_ERR_READ	читать старейшую ошибку выполнения

Возможные аргументы предопределенных функций функции SYSTEM:

Команда	Аргумент	Возвращаемая величина
SYS_TALLOWED	0	допустимое время цикла
SYS_TCURRENT	0	текущее время цикла
SYS_TMAXIMUM	0	максимальное время цикла
SYS_TOVERFLOW	0	количество переполнений времени цикла
SYS_TRESET	0	0
SYS_TWRITE	новое время цикла	записанное время цикла
SYS_ERR_TEST	0	0 если ошибок нет
SYS_ERR_READ	0	код старейшей ошибки выполнения

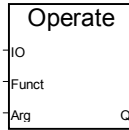
(\*FBD пример блоков SYSTEM\*)



(\* STэквивалент: \*)

```
alarm := (SYSTEM (SYS_TOVERFLOW, 0) <> 0);
If (alarm) Then
    nb_err := nb_err + 1;
    rc := SYSTEM (SYS_TRESET, 0);
End_If;
```

## OPERATE



Аргументы:

<b>IO</b>	ANY	вход или выход
<b>Funct</b>	INT	управляемое действие
<b>Arg</b>	INT	аргумент действия В/В
<b>Q</b>	INT	возврат проверки

Описание:

Доступ к каналам В/В.

Значение аргументов блока OPERATE зависит от реализации интерфейса В/В. См. Описание соответствующих плат В/В.

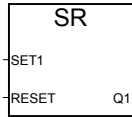
### В.9.2 Стандартные функциональные блоки

Существуют следующие стандартные функциональные блоки, поддерживаемые системой ISaGRAF. Эти функциональные блоки предопределены и их не нужно заявлять в библиотеке.

Булевские.....	SR	установить доминанту
	RS	сбросить доминанту
	R_Trig	определение переднего фронта
	F_Trig	определение заднего фронта
	SEMA	семафор
Счетчики.....	CTU	счетчик вверх
	CTD	счетчик вниз
	CTUD	счетчик вверх-вниз
Таймеры.....	TON	время включения
	TOF	время выключения
	TP	время пульсирования
Целые аналоги.....	CMP	сравнение
	StackInt	Стек целых аналоговых
Действительные аналоги.....	AVERAGE	Среднее из N образцов
	HYSTER	Булевский гистерезис на разнице действительных
	LIM_ALRM	Тревога верхнего/нижнего предела с гистерезисом
	INTEGRAL	Интегрирование по времени
	DERIVATE	Дифференцирование по времени
Генерация сигналов.....	BLINK	Мерцающий булевский сигнал
	SIG_GEN	Генератор сигнала

Замечание: Если созданы новые функциональные блоки на "С", они могут быть вызваны из языка FBD.

## SR



Аргументы:

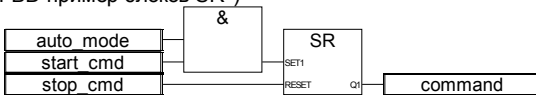
<b>SET1</b>	BOO	если TRUE, установить Q1 в TRUE (доминанта)
<b>RESET</b>	BOO	если TRUE, сбросить Q1 в FALSE
<b>Q1</b>	BOO	булевское состояние в памяти

Описание:

Устанавливает доминанту. См. Таблицу:

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

(\*FBD пример блоков SR\*)



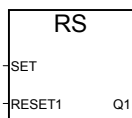
(\* ST эквивалент: Мы предполагаем, что SR1 - это экземпляр блока SR \*)

```
SR1((auto_mode & start_cmd), stop_cmd);
command := SR1.Q1;
```

(\* IL Эквивалент: \*)

```
LD      auto_mode
AND     start_cmd
ST      SR1.set1
LD      stop_cmd
ST      SR1.reset
CAL     SR1
LD      SR1.Q1
ST      command
```

## RS



Аргументы:

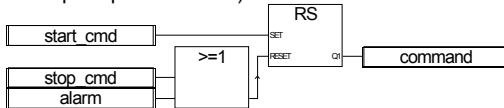
<b>SET</b>	BOO	если TRUE, установить Q1 в TRUE
<b>RESET1</b>	BOO	если TRUE, установить Q1 в FALSE (доминанта)
<b>Q1</b>	BOO	булевское состояние в памяти

Описание:

Сбрасывает доминанту. См. Таблицу:

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

(\*FBD пример блоков RS\*)

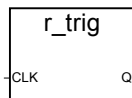


(\* ST Эквивалент: Мы предполагаем, что RS1 - это экземпляр блока RS \*)  
RS1(start\_cmd, (stop\_cmd OR alarm));  
command := RS1.Q1;

(\* IL Эквивалент: \*)

```
LD      start_cmd
ST      RS1.set
LD      stop_cmd
OR      alarm
ST      RS1.reset1
CAL     RS1
LD      RS1.Q1
ST      command
```

## R\_TRIG



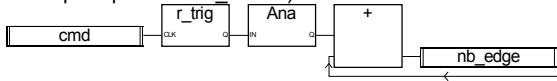
Аргументы:

<b>CLK</b>	BOO	любая булевская переменная
<b>Q</b>	BOO	TRUE, когда CLK поднимается с FALSE на TRUE
	TRUE	FALSE во всех остальных случаях

Описание:

Определяет передний фронт булевской переменной.

(\*FBD пример блоков R\_TRIG\*)



(\* ST Эквивалент: Мы предполагаем, что R\_TRIG1 - это экземпляр блока R\_TRIG \*)

R\_TRIG1(cmd);

nb\_edge := ANA(R\_TRIG1.Q) + nb\_edge;

(\* ILЭквивалент: \*)

LD cmd

ST R\_TRIG1.clk

CAL R\_TRIG1

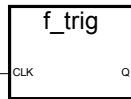
LD R\_TRIG1.Q

ANA

ADD nb\_edge

ST nb\_edge

## F\_TRIG



Аргументы:

**CLK**

**Q**

BOO любая булевская переменная

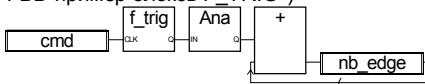
BOO TRUE, когда CLK меняется с TRUE на FALSE

FALSE во всех остальных случаях

Описание:

Определяет задний фронт булевской переменной.

(\*FBD пример блоков F\_TRIG\*)



(\* ST эквивалент: Мы предполагаем, что F\_TRIG1 – экземпляр блока F\_TRIG \*)

F\_TRIG1(cmd);

nb\_edge := ANA(F\_TRIG1.Q) + nb\_edge;

(\* IL Эквивалент: \*)

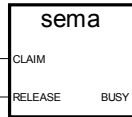
LD cmd

ST F\_TRIG1.clk

CAL F\_TRIG1

LD F\_TRIG1.Q  
 ANA  
 ADD nb\_edge  
 ST nb\_edge

## SEMA



Аргументы:

<b>CLAIM</b>	BOOLEAN	команда "проверить и установить"
<b>RELEASE</b>	BOOLEAN	освободить семафор
<b>BUSY</b>	BOOLEAN	состояние семафора

Описание:

(\* "x" - это булевская переменная проинициализированная значением FALSE \*)

busy := x;

If claim Then

    x := True;

Else

    If release Then

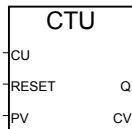
        busy := False;

        x := False;

    End\_if;

End\_if;

## CTU



Аргументы:

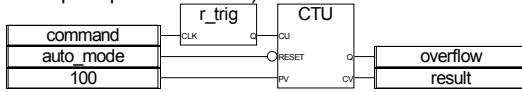
<b>CU</b>	BOO	вход для счета (считать когда CU равно TRUE)
<b>RESET</b>	BOO	команда сброса (доминанта)
<b>PV</b>	INT	планируемое максимальное значение
<b>Q</b>	BOO	переполнение: TRUE если CV = PV
<b>CV</b>	INT	результат счета

Предупреждение: Блок CTU не определяет передний и задний фронты входа (CU). Для того чтобы создать импульсный счетчик, его нужно связать с блоками "R\_TRIG" или "F\_TRIG".

Описание:

Считает от 0 до заданного значения по 1.

(\*FBD пример блоков CTU\*)



(\* ST Эквивалент: Мы предполагаем, что R\_TRIG1 - это экземпляр блока R\_TRIG и CTU1 - это экземпляр блока CTU \*)

CTU1(R\_TRIG1(command),NOT(auto\_mode),100);

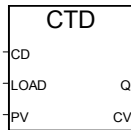
overflow := CTU1.Q;

result := CTU1.CV;

(\* IL Эквивалент: \*)

```
LD      command
ST      R_TRIG1.clk
CAL    R_TRIG1
LD      R_TRIG1.Q
ST      CTU1.cu
ST      CTU1.reset
LDN    auto_mode
ST      CTU1.reset
LD      100
ST      CTU1.pv
CAL    CTU1
LD      CTU1.Q
ST      overflow
LD      CTU1.cv
ST      result
```

## CTD



Аргументы:

**CD**

BOO вход для счета (считать вниз когда CD равно TRUE)

**LOAD**

(счет вниз если CD равно TRUE)  
BOO команда загрузить (доминанта)  
(CV = PV если LOAD равно TRUE)

**PV**

INT планируемое максимальное значение

**Q**

BOO переполнение: TRUE если CV = 0

**CV**

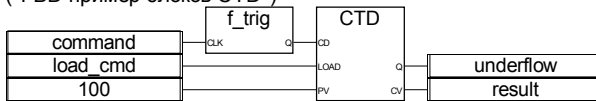
INT результат счета

Предупреждение: Блок CTD не определяет передний и задний фронты входа (CD). Для того чтобы создать импульсный счетчик, его нужно связать с блоками "R\_TRIG" или "F\_TRIG".

Описание:

Считает от заданного значения вниз до 0 по 1.

(\*FBD пример блоков CTD\*)



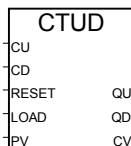
(\* ST Эквивалент: Мы предполагаем, что R\_TRIG1 - это экземпляр блока R\_TRIG и CTD1 - это экземпляр блока CTD \*)

```
CTD1(F_TRIG1(command),load_cmd,100);
underflow := CTD1.Q;
result := CTD1.CV;
```

(\* IL Эквивалент: \*)

```
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



Аргументы:

<b>CU</b>	BOO	вход для счета вверх (считать когда CU равно TRUE)
<b>CD</b>	BOO	вход для счета вниз (считать когда CD равно TRUE)
<b>RESET</b>	BOO	команда сброса (доминанта) (CV=0 когда RESET = TRUE)
<b>LOAD</b>	BOO	команда загрузить (CV=PV когда LOAD = TRUE)
<b>PV</b>	INT	планируемое максимальное значение
<b>QU</b>	BOO	переполнение: TRUE если CV = PV
<b>QD</b>	BOO	переполнение: TRUE если CV = 0
<b>CV</b>	INT	результат счета

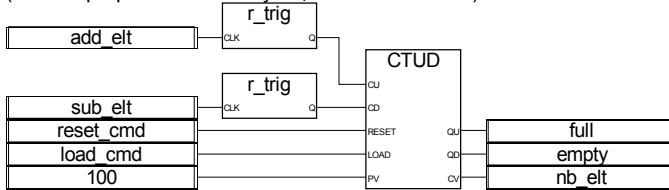
Предупреждение: Блок CTUD не определяет передний и задний фронты входа (CU и CD). Для того чтобы создать импульсный счетчик, его нужно связать с блоками "R\_TRIG" или "F\_TRIG".



Описание:

Считает от 0 до заданного значения по 1 или от заданного значения вниз до 0 по 1.

(\* FBD программа использующая блок "CTUD" \*)



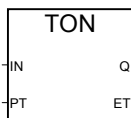
(\* ST Эквивалент: Мы предполагаем, что R\_TRIG1 и R\_TRIG2 - это экземпляры блока R\_TRIG и CTD1 - это экземпляр блока CTUD \*)

```
CTUD1(R_TRIG1(add_elt), R_TRIG2(sub_elt), reset_cmd, load_cmd,100);
full := CTUD1.QU;
empty := CTUD1.QD;
nb_elt := CTUD1.CV;
```

(\* IL Эквивалент: \*)

```
LD      add_elt
ST      R_TRIG1.clk
CAL     R_TRIG1
LD      R_TRIG1.Q
ST      CTUD1.cu
LD      sub_elt
ST      R_TRIG2.clk
CAL     R_TRIG2
LD      R_TRIG2.Q
ST      CTUD1.cd
LD      reset_cmd
ST      CTUD1.reset
LD      load_cmd
ST      CTUD1.load
LD      100
ST      CTUD1.pv
CAL     CTUD1
LD      CTUD1.QU
ST      full
LD      CTUD1.QD
ST      empty
LD      CTUD1.CV
ST      nb_elt
```

**TON**



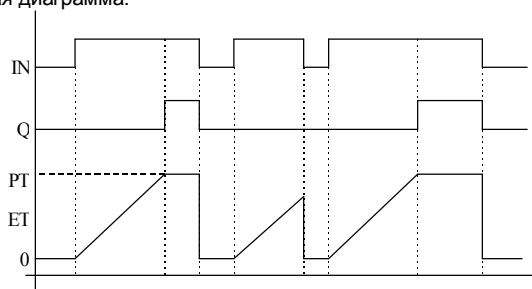
Аргументы:

<b>IN</b>	BOO	если передний фронт, то начинает увеличивать внутренний таймер, если задний фронт, то останавливается и сбрасывает внутренний таймер
<b>PT</b>	TMR	максимальное планируемое время
<b>Q</b>	BOO	если TRUE, то планируемое время истекло
<b>ET</b>	TMR	текущее прошедшее время

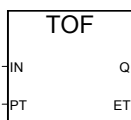
Описание:

Увеличивать внутренний таймер до заданной величины.

Временная диаграмма:



## TOF



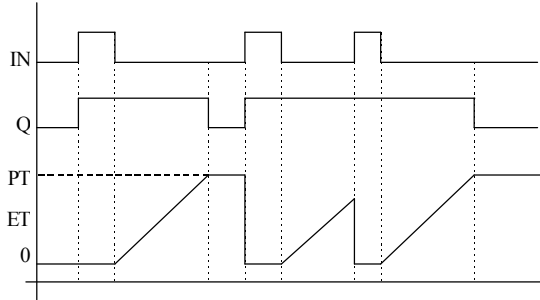
Аргументы:

<b>IN</b>	BOO	если задний фронт, то начинает увеличивать внутренний таймер если передний фронт, то останавливается и сбрасывает внутренний таймер
<b>PT</b>	TMR	максимальное планируемое время
<b>Q</b>	BOO	если TRUE, то планируемое время истекло
<b>ET</b>	TMR	текущее прошедшее время

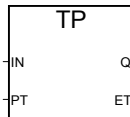
Описание:

Увеличивать внутренний таймер до заданной величины.

Временная диаграмма:



## TP



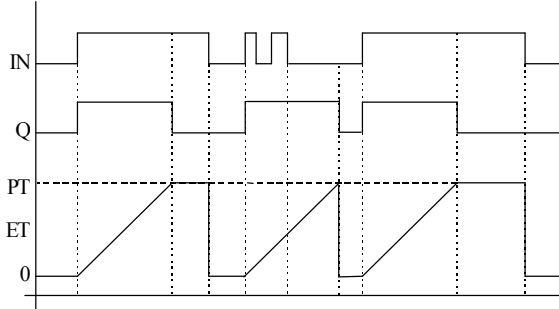
Arguments:

<b>IN</b>	BOO	если передний фронт, то начинает увеличивать внутренний таймер (если уже не увеличивается) если задний фронт и время истекло, то сбрасывает внутренний таймер
<b>PT</b>	TMR	любое изменение IN во время счета не имеет эффекта. максимальное планируемое время
<b>Q</b>	BOO	если TRUE, то таймер считает
<b>ET</b>	TMR	текущее прошедшее время

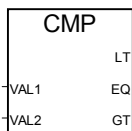
Описание:

Увеличивать внутренний таймер до заданной величины.

Временная диаграмма:



## CMP



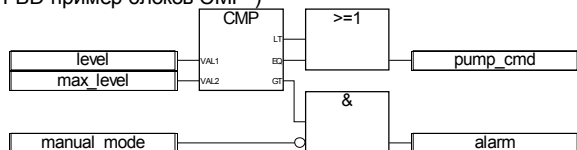
Аргументы:

<b>VAL1</b>	INT	любое знаковое целое аналоговое значение
<b>VAL2</b>	INT	любое знаковое целое аналоговое значение
<b>LT</b>	BOO	TRUE если val1 меньше чем val2
<b>EQ</b>	BOO	TRUE если val1 равно val2
<b>GT</b>	BOO	TRUE если val1 больше чем val2

Описание:

Сравнивает две величины: сообщает они равны или первая больше или меньше второй.

(\*FBD пример блоков CMP\*)



(\* ST Эквивалент: Мы предполагаем, что CMP1 - это экземпляр блока CMP \*)

```

CMP1(level, max_level);
pump_cmd := CMP1.LT OR CMP1.EQ;
alarm := CMP1.GT AND NOT(manual_mode);

```

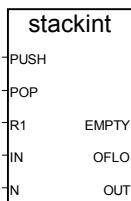
(\* IL Эквивалент: \*)

```

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



Аргументы:

<b>PUSH</b>	BOO	команда "PUSH" (на переднем фронте) добавить значение IN в верхушку стека
<b>POP</b>	BOO	команда "POP" (на переднем фронте) удалить из верхушки стека последнее сохраненное значение
<b>R1</b>	BOO	сбросить стек в пустое состояние
<b>IN</b>	INT	сохраняемое значение
<b>N</b>	INT	размер стека определенный приложением
<b>EMPTY</b>	BOO	TRUE если стек пустой
<b>OFLO</b>	BOO	переполнение: TRUE если стек переполнен
<b>OUT</b>	INT	значение в верхушке стека

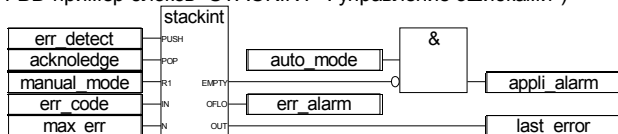
Описание:

Управляет стеком целых значений.

Функциональный блок "STACKINT" включает определение переднего фронта для команд PUSH и POP. Максимальный размер стека 128. Размер стека определенный приложением - N не может быть меньше 1 и больше 128.

Заметим, что значение OFLO действительно только после сброса (R1 было установлено в TRUE, по крайней мере, однажды и обратно в FALSE)

(\*FBD пример блоков "STACKINT" : управление ошибками\*)



(\* ST Эквивалент: Мы предполагаем, что STACKINT1 - это экземпляр блока STACKINT \*)

STACKINT1(err\_detect, acknowledge, manual\_mode, err\_code, max\_err);

appli\_alarm := auto\_mode AND NOT(STACKINT1.EMPTY);

err\_alarm := STACKINT1.OFLO;

last\_error := STACKINT1.OUT;

(\* IL Эквивалент: \*)

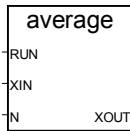
```
LD      err_detect
ST      STACKINT1.push
LD      acknowledge
ST      STACKINT1.pop
LD      manual_mode
ST      STACKINT1.r1
```

```

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**



Аргументы:

<b>RUN</b>	BOO	TRUE=запустить/FALSE=сбросить
<b>XIN</b>	REAL	любая аналоговая переменная
<b>N</b>	INT	количество образцов, заданное приложением
<b>XOUT</b>	REAL	среднее значение XIN

Описание:

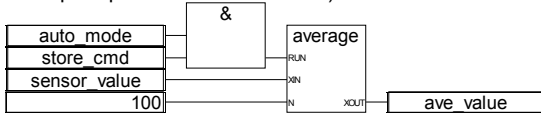
Запоминает значение на каждом цикле и вычисляет среднее значение всех уже запомненных величин. Запоминает только N последних значений.

Количество образцов не может превосходить 128.

Если команда "RUN" равна FALSE, то значение на выходе равно значению на входе.

Когда достигнуто максимальное значение N сохраненных величин, первое запомненное значение вытесняется последним.

(\*FBD пример блоков "AVERAGE" \*)



(\* ST Эквивалент: Мы предполагаем, что AVERAGE1 - это экземпляр блока AVERAGE \*)

```
AVERAGE1((auto_mode & store_cmd), sensor_value, 100);
```

```
ave_value := AVERAGE1.XOUT;
```

(\* IL Эквивалент: \*)

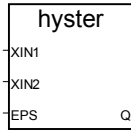
```

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



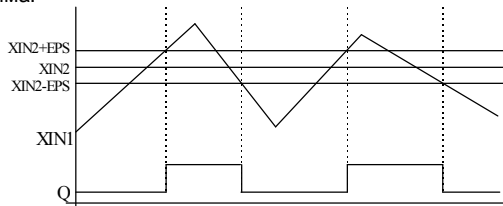
Аргументы:

<b>XIN</b>	REAL	любое аналоговое значение
<b>XIN2</b>	REAL	любое аналоговое значение
<b>EPS</b>	REAL	величина гистерезиса (должна быть больше нуля)
<b>Q</b>	BOO	TRUE если XIN1 перешел через XIN2 + EPS, но Еще не ниже XIN2 - EPS

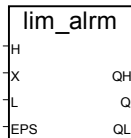
Описание:

Гистерезис действительного значения по верхнему пределу.

Временная диаграмма:



## LIM\_ALARM



Аргументы:

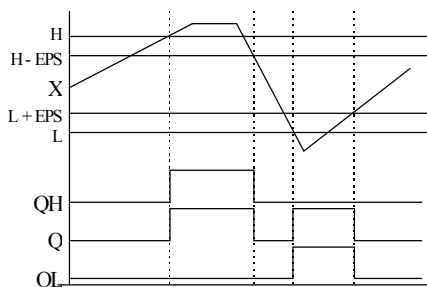
<b>H</b>	REAL	значение верхнего предела
<b>X</b>	REAL	вход: любое аналоговое значение
<b>L</b>	REAL	значение нижнего предела
<b>EPS</b>	REAL	величина гистерезиса (должна быть больше нуля)
<b>QH</b>	BOO	"верхняя" тревога: TRUE если X выше верхнего предела H

<b>Q</b>	BOO	тревога: TRUE если X вне пределов
<b>QL</b>	BOO	“нижняя” тревога: TRUE если X ниже нижнего предела L

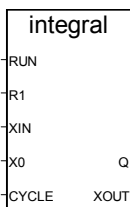
Описание:

Гистерезис действительного значения по верхнему и нижнему пределам. Гистерезис налагается на верхний и на нижний пределы. Дельта гистерезиса, используемая, как для нижнего предела так и для верхнего, равна половине параметра EPS.

Временная диаграмма:



## INTEGRAL



Аргументы:

<b>RUN</b>	BOO	режим: TRUE=интегрировать/FALSE=держать
<b>R1</b>	BOO	сброс
<b>XIN</b>	REAL	вход: любая действительная аналоговая величина
<b>X0</b>	REAL	начальное значение
<b>CYCLE</b>	TMR	период интегрирования
<b>Q</b>	BOO	не R1
<b>XOUT</b>	REAL	интегрированный выход

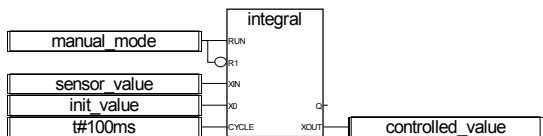
Описание:

Интеграл действительной величины

Если величина параметра CYCLE меньше чем время цикла приложения ISaGRAF, то период интегрирования будет равен времени цикла приложения ISaGRAF.

(\*FBD пример блока "INTEGRAL" \*)



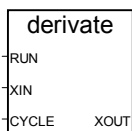


(\* ST Эквивалент: Мы предполагаем, что INTEGRAL1 - это экземпляр блока INTEGRAL \*)  
 INTEGRAL1(manual\_mode, NOT(manual\_mode), sensor\_value, init\_value, t#100ms);  
 controlled\_value := INTEGRAL1.XOUT;

(\* IL Эквивалент: \*)

```
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



Аргументы:

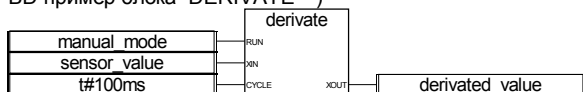
<b>RUN</b>	BOO	режим: TRUE=нормальный/FALSE=сброс
<b>XIN</b>	REAL	вход: любая действительная аналоговая величина
<b>CYCLE</b>	TMR	период дифференцирования
<b>XOUT</b>	REAL	интегрированный выход

Описание:

Дифференцирование действительной величины

Если величина параметра CYCLE меньше чем время цикла приложения ISaGRAF, то период дифференцирования будет равен времени цикла приложения ISaGRAF.

(\*FBD пример блока "DERIVATE" \*)



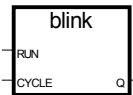
(\* ST Эквивалент: Мы предполагаем, что DERIVATE1 - это экземпляр блока DERIVATE \*)

```
DERIVATE1(manual_mode, sensor_value, t#100ms);
derivated_value := DERIVATE1.XOUT;
```

(\* IL Эквивалент: \*)

```
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



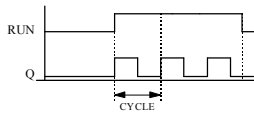
Аргументы:

<b>RUN</b>	BOO	режим : TRUE=мигать/FALSE=сброс выхода в FALSE
<b>CYCLE</b>	TMR	период мигания
<b>Q</b>	BOO	выходной мигающий сигнал

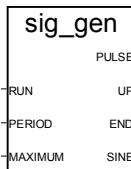
Описание:

Генерирует мигающий сигнал.

Временная диаграмма:



## SIG\_GEN



Аргументы:

<b>RUN</b>	BOO	режим : TRUE=работать/FALSE=сброс выхода в FALSE
<b>PERIOD</b>	TMR	период образца
<b>MAXIMUM</b>	INT	максимальное значение

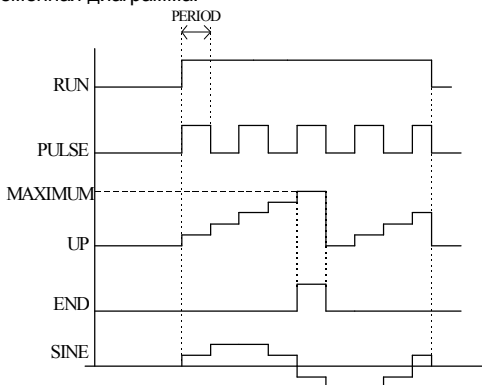
<b>PULSE</b>	BOO	инвертируется после каждого образца
<b>UP</b>	INT	счетчик, увеличивающийся на каждом образце
<b>END</b>	BOO	TRUE, когда заканчивается счет вверх
<b>SINE</b>	REAL	синусоидальный сигнал (период = время счета)

Описание:

Генерирует различные сигналы : булевский мигающий, целый счетчик вверх, действительная синусоида.

Когда счетчик достигает максимума, он перезапускается с нуля. Так что END удерживает TRUE в течении одного периода.

Временная диаграмма:



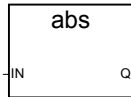
### В.9.3 Стандартные функции

Существуют следующие стандартные функции, поддерживаемые системой ISaGRAF. Эти функции предопределены и их не нужно заявлять в библиотеке.

Математические.....	ABS	модуль
	EXPT, POW	экспонента, степень
	LOG	логарифм
	SQRT	корень квадратный
	TRUNC	отрезать действительную часть
Тригонометрические.....	ACOS, ASIN	арккосинус, арксинус
	ATAN	арктангенс
	COS, SIN	косинус, синус
	TAN	тангенс
Работа с регистрами .....	ROL, ROR	вращать влево, вращать вправо
	SHL, SHR	сдвиг влево, сдвиг вправо
Работа с данными .....	MIN, MAX	Минимум, Максимум
	LIMIT	Ограничение
	MOD	Модуль
	MUX4, MUX8	Мультиплексор (4 и 8 входов)
	SEL	Двоичный селектор
	ODD	Четность

Превращение данных.....	RAND	Случайная величина
	ASCII	Символ à ASCII код
	CHAR	ASCII код à символ
Работа со строками.....	MLEN	Взять длину строки
	DELETE	Удалить подстроку
	INSERT	Вставить строку
	FIND	Найти подстроку
	REPLACE	Заменить подстроку
	LEFT, MID	Извлечь левую часть, середину
	RIGHT	Извлечь правую часть
	DAY_TIME	Время дня
Работа с массивами .....	ARCREATE	Создать массив целых чисел
	ARREAD	Читать /
	ARWRITE	Писать элемент массива
Работа с двоичным файлом ....	F_ROPEN	Открыть двоичный файл в режиме чтения
	F_WOPEN	Открыть двоичный файл в режиме записи
	F_CLOSE	Закрыть двоичный файл
	F_EOF	Проверить конец двоичного файла
	FA_READ	Читать из двоичного файла
	FA_WRITE	Записать в двоичный файл
	FM_READ	Читать из двоичного файла строку
	FM_WRITE	Записать в двоичный файл строку

## ABS



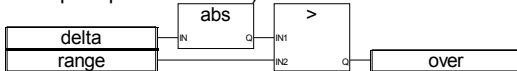
Аргументы:

<b>IN</b>	REAL	любая знаковая аналоговая величина
<b>Q</b>	REAL	модуль аналоговой величины

Описание:

Дает абсолютное значение действительной величины.

(\*FBD пример блока "ABS"\*)



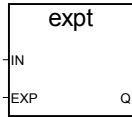
(\* ST Эквивалент: \*)

over := (ABS (delta) > range);

(\* IL Эквивалент: \*)

LD	delta
ABS	
GT	range
ST	over

## EXPT



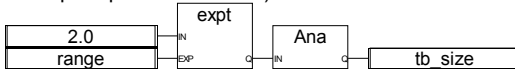
Аргументы:

<b>IN</b>	REAL	любая знаковая аналоговая величина
<b>EXP</b>	INT	целая аналоговая экспонента
<b>Q</b>	REAL	( $IN^{EXP}$ )

Описание:

Дает действительный результат операции: (база <sup>экспонента</sup>) 'база' - первый аргумент, экспонента - второй.

(\*FBD пример блока "EXPT"\*)



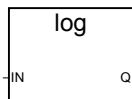
(\* ST Эквивалент: \*)

tb\_size := ANA (EXPT (2.0, range) );

(\* IL Эквивалент: \*)

```
LD      2.0
EXPT   range
ANA
ST      tb_size
```

## LOG



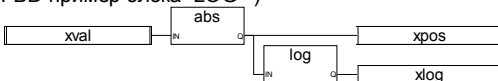
Аргументы:

<b>IN</b>	REAL	аналоговая величина должна быть больше нуля
<b>Q</b>	REAL	логарифм (основание 10) входа

Описание:

Вычисляет логарифм (основание 10) действительной величины.

(\*FBD пример блока "LOG"\*)



(\* ST Эквивалент: \*)

xpos := ABS (xval);

xlog := LOG (xpos);

(\* IL Эквивалент: \*)  
LD xval  
ABS  
ST xpos  
LOG  
ST xlog

## POW



Аргументы:

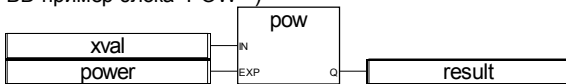
<b>IN</b>	REAL	любая знаковая аналоговая величина
<b>EXP</b>	REAL	степень
<b>Q</b>	REAL	( $IN^{EXP}$ )

1.0 если IN не 0.0 и EXP = 0.0  
0.0 если IN = 0.0 и EXP отрицательно  
0.0 если IN = 0ю0 и EXP = 0.0  
0.0 если IN отрицательно и EXP не целое

Описание:

Дает действительный результат операции: (база <sup>экспонента</sup>) 'база' - первый аргумент, экспонента - второй. Экспонента - действительное число.

(\*FBD пример блока "POW"\*)



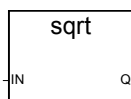
(\* ST Эквивалент: \*)

result := POW (xval, power);

(\* IL Эквивалент: \*)

LD xval  
POW power  
ST result

## SQRT



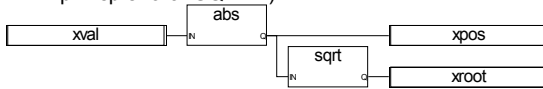
Аргументы:

<b>IN</b>	REAL	аналоговая величина должна быть больше нуля
<b>Q</b>	REAL	квадратный корень входа

Описание:

Вычисляет квадратный корень действительной величины.

(\*FBD пример блока "SQRT"\*)



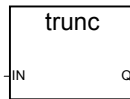
(\* ST Эквивалент: \*)

```
xpos := ABS (xval);  
xroot := SQRT (xpos);
```

(\* IL Эквивалент: \*)

```
LD      xval  
ABS  
ST      xpos  
SQRT  
ST      xroot
```

## TRUNC



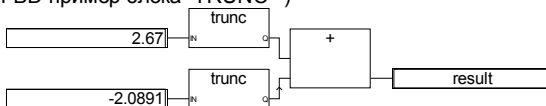
Аргументы:

<b>IN</b>	REAL	любая действительная аналоговая величина
<b>Q</b>	REAL	если IN>0 наибольшее целое меньшее или равное входа если IN<0 наименьшее целое большее или равное входа

Описание:

Отсекает дробную часть.

(\*FBD пример блока "TRUNC"\*)



(\* ST Эквивалент: \*)

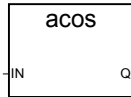
```
result := TRUNC (+2.67) + TRUNC (-2.0891);  
(* означает: result := 2.0 + (-2.0) := 0.0; *)
```

(\* IL Эквивалент: \*)

```
LD      2.67
```

TRUNC  
 ST temporary (\* временный результат TRUNC \*)  
 LD -2.0891  
 TRUNC  
 ADD temporary  
 ST result

## ACOS



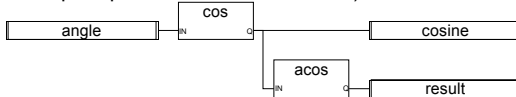
Аргументы:

<b>IN</b>	REAL	должен быть в диапазоне [-1.0..+1.0]
<b>Q</b>	REAL	арккосинус входа в диапазоне [0.0..Pi] 0.0 для неверного входа

Описание:

Вычисляет арккосинус действительной величины.

(\*FBD пример блока "COS" и "ACOS"\*)



(\* ST Эквивалент: \*)

cosine := COS (angle);

result := ACOS (cosine); (\*результат равен углу \*)

(\* IL Эквивалент: \*)

LD angle

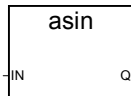
COS

ST cosine

ACOS

ST result

## ASIN



Аргументы:

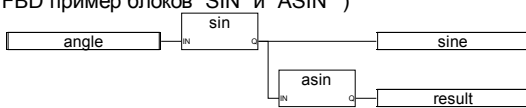
<b>IN</b>	REAL	должен быть в диапазоне [-1.0..+1.0]
<b>Q</b>	REAL	арксинус входа в диапазоне [-Pi/2..+Pi/2] 0.0 для неверного входа

Описание:

Вычисляет арксинус действительной величины.



(\*FBD пример блоков "SIN" и "ASIN"\*)



(\* ST Эквивалент: \*)

sine := SIN (angle);

result := ASIN (sine); (\*результат равен углу \*)

(\* IL Эквивалент: \*)

LD angle

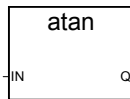
SIN

ST sine

ASIN

ST result

## ATAN



Аргументы:

**IN**

REAL

любая действительная аналоговая

**Q**

REAL

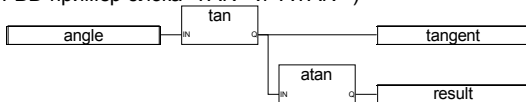
арктангенс входа в диапазоне  $[-\pi/2..+\pi/2]$

0.0 для неверного входа

Описание:

Вычисляет арктангенс действительной величины.

(\*FBD пример блока "TAN" и "ATAN"\*)



(\* ST Эквивалент: \*)

tangent := TAN (angle);

result := ATAN (tangent); (\*результат равен углу \*)

(\* IL Эквивалент: \*)

LD angle

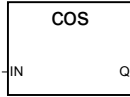
TAN

ST tangent

ATAN

ST result

## COS



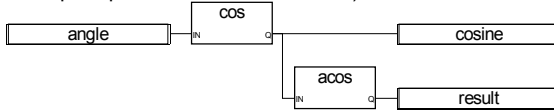
Аргументы:

**IN** REAL любая действительная аналоговая  
**Q** REAL косинус входа в диапазоне [-1.0..+1.0]

Описание:

Вычисляет косинус действительной величины.

(\*FBD пример блока "COS" и "ACOS"\*)



(\* ST Эквивалент: \*)

cosine := COS (angle);

result := ACOS (cosine); (\*результат равен углу \*)

(\* IL Эквивалент: \*)

LD angle

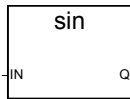
COS

ST cosine

ACOS

ST result

## SIN



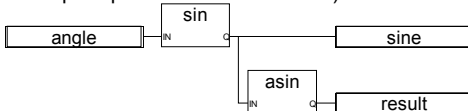
Аргументы:

**IN** REAL любая действительная аналоговая  
**Q** REAL синус входа в диапазоне [-1.0..+1.0]

Описание:

Вычисляет синус действительной величины.

(\*FBD пример блока "SIN" и "ASIN"\*)

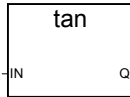


(\* ST Эквивалент: \*)

sine := SIN (angle);  
 result := ASIN (sine); (\*результат равен углу \*)

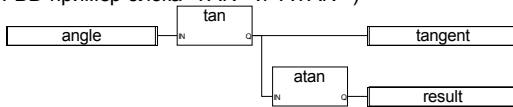
(\* IL Эквивалент: \*)  
 LD angle  
 SIN  
 ST sine  
 ASIN  
 ST result

## TAN



Аргументы:  
**IN** REAL не может равняться PI/2 по модулю PI  
**Q** REAL тангенс входа  
 =1У+38 для неверного входа

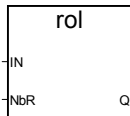
Описание:  
 Вычисляет тангенс действительной величины.  
 (\*FBD пример блока "TAN" и "ATAN"\*)



(\* ST Эквивалент : \*)  
 tangent := TAN (angle);  
 result := ATAN (tangent); (\*результат равен углу \*)

(\* IL Эквивалент: \*)  
 LD angle  
 TAN  
 ST tangent  
 ATAN  
 ST result

## ROL

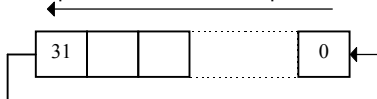


Аргументы:  
**IN** INT любая целая аналоговая величина

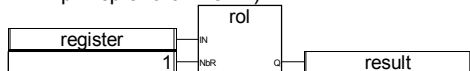
**NbR** INT количества вращаемых бит (в диапазоне [1..31])  
**Q** INT повернутая влево величина  
если NbR <= 0, то нет никакого эффекта

Описание:

Вращает биты влево. Вращаются 32 бита.



(\*FBD пример блока "ROL"\*)



(\* ST Эквивалент: \*)

result := ROL (register, 1);

(\* register = 2#0100\_1101\_0011\_0101\*)

(\* result = 2#1001\_1010\_0110\_1010\*)

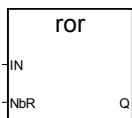
(\* IL Эквивалент: \*)

LD register

ROL 1

ST result

## ROR

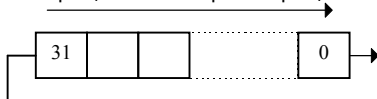


Аргументы:

**IN** INT любая целая аналоговая величина  
**NbR** INT количества вращаемых бит (в диапазоне [1..31])  
**Q** INT повернутая вправо величина  
если NbR <= 0, то нет никакого эффекта

Описание:

Вращает биты вправо. Вращаются 32 бита:



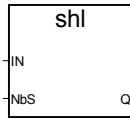
(\*FBD пример блока "ROR"\*)



```
(* ST Эквивалент: *)
result := ROR (register, 1);
(* register = 2#0100_1101_0011_0101 *)
(* result = 2#1010_0110_1001_1010 *)
```

```
(* IL : *)
LD      register
ROR     1
ST      result
```

## SHL

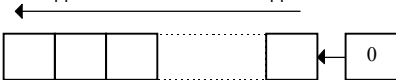


Аргументы:

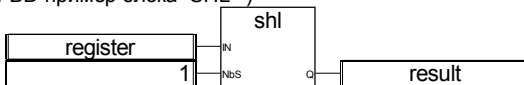
<b>IN</b>	INT	любая целая аналоговая величина
<b>NbR</b>	INT	количества сдвигаемых бит (в диапазоне [1..31])
<b>Q</b>	INT	провернутая вправо величина если NbR <= 0, то нет никакого эффекта младший бит заменяет нулем

Описание:

Сдвигает биты влево. Сдвигаются 32 бита.



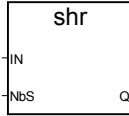
(\*FBD пример блока "SHL"\*)



```
(* ST Эквивалент: *)
result := SHL (register,1);
(* register = 2#0100_1101_0011_0101 *)
(* result = 2#1001_1010_0110_1010 *)
```

```
(* IL Эквивалент: *)
LD      register
SHL     1
ST      result
```

## SHR

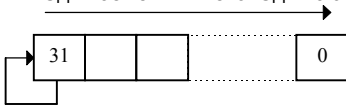


Аргументы:

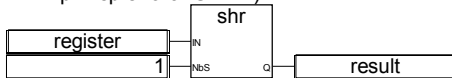
<b>IN</b>	INT	любая целая аналоговая величина
<b>NbR</b>	INT	количества сдвигаемых бит (в диапазоне [1..31])
<b>Q</b>	INT	провернутая вправо величина если NbR <= 0, то нет никакого эффекта младший бит заменяет нулем

Описание:

Сдвигает биты влево. Сдвигаются 32 бита.



(\*FBD пример блока "SHR"\*)



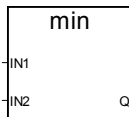
(\* ST Эквивалент: \*)

```
result := SHR (register,1);
(* register = 2#1100_1101_0011_0101 *)
(* result   = 2#1110_0110_1001_1010 *)
```

(\* IL Эквивалент: \*)

```
LD     register
SHR    1
ST     result
```

## MIN



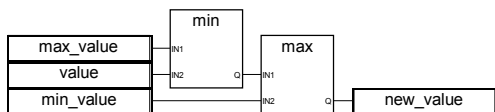
Аргументы:

<b>IN</b>	INT	любая знаковая целая величина
<b>IN2</b>	INT	(не может быть REAL)
<b>Q</b>	INT	минимум из двух входов

Описание:

Определяет минимальное значение из двух целых.

(\*FBD пример блоков "MIN" и "MAX"\*)



(\* ST Эквивалент: \*)

$new\_value := MAX (MIN (max\_value, value), min\_value);$

(\*ограничивает значение в диапазоне [min\_value..max\_value] \*)

(\* IL Эквивалент: \*)

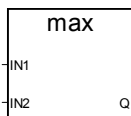
LD max\_value

MIN value

MAX min\_value

ST new\_value

## MAX



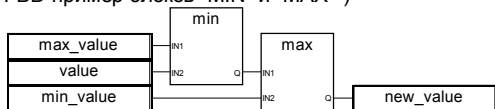
Аргументы:

<b>IN1</b>	INT	любая знаковая целая величина
<b>IN2</b>	INT	(не может быть REAL)
<b>Q</b>	INT	максимум из двух входов

Описание:

Определяет максимальное значение из двух целых.

(\*FBD пример блоков "MIN" и "MAX"\*)



(\* ST Эквивалент: \*)

$new\_value := MAX (MIN (max\_value, value), min\_value);$

(\*ограничивает значение в диапазоне [min\_value..max\_value] \*)

(\* IL Эквивалент: \*)

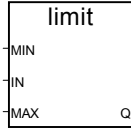
LD max\_value

MIN value

MAX min\_value

ST new\_value

## LIMIT



Аргументы:

<b>MIN</b>	INT	минимальная допустимая величина
<b>IN</b>	INT	любая знаковая целая величина
<b>MAX</b>	INT	максимальная допустимая величина
<b>Q</b>	INT	выходная величина ограниченная заданным диапазоном

Описание:

Удерживает величину в заданном диапазоне. Величина сохраняется если она между максимумом и минимумом или становится равной максимуму если она больше максимума и становится равной минимуму если она меньше минимума.

(\*FBD пример блока "LIMIT"\*)



(\* ST Эквивалент: \*)

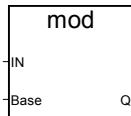
`new_value := LIMIT (min_value, value, max_value);`

(\* ограничивает значение в диапазоне [min\_value..max\_value] \*)

(\* IL Эквивалент: \*)

```
LD      min_value
LIMIT  value, max_value
ST      new_value
```

## MOD



Аргументы:

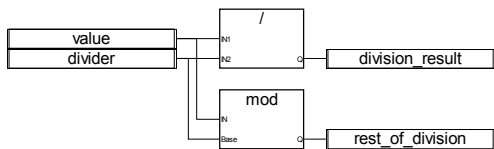
<b>IN</b>	INT	любая знаковая целая величина
<b>Base</b>	INT	должна быть больше нуля
<b>Q</b>	INT	вычисление модуля (вход MOD база) возвращает -1 если Base <= 0

Описание:

Вычисляет модуль целого значения.

(\*FBD пример блока "MOD"\*)





(\* ST Эквивалент: \*)

division\_result := (value / divider); (\* целое деление \*)

rest\_of\_division := MOD (value, divider); (\* остаток от деления \*)

(\* IL Эквивалент: \*)

LD value

DIV divider

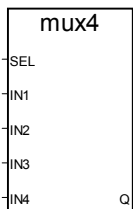
ST division\_result

LD value

MOD divider

ST rest\_of\_division

## MUX4



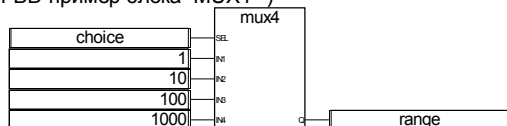
Аргументы:

<b>SEL</b>	INT	целый селектор (должен быть в диапазоне [0..3])
<b>IN1..IN4</b>	INT	целая аналоговая величина
<b>Q</b>	INT	= value1 if SEL = 0 = value2 if SEL = 1 = value3 if SEL = 2 = value4 if SEL = 3 = 0 для других значений селектора

Описание:

Мультиплексор 4 входов: выбирает одно из четырех целых чисел.

(\*FBD пример блока "MUX4"\*)



(\* ST Эквивалент: \*)

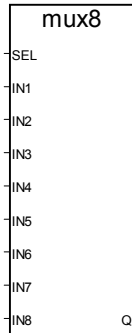
range := MUX4 (choice, 1, 10, 100, 1000);

(\* выбор из 4 predeterminedенных значений, например, если выбор - 1, значение будет 10 \*)

(\* IL Эквивалент: \*)

LD choice  
MUX4 1,10,100,1000  
ST range

## MUX8



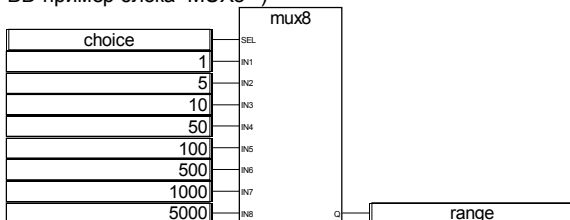
Аргументы:

<b>SEL</b>	INT	целый селектор (должен быть в диапазоне [0..7])
<b>IN1..IN8</b>	INT	целая аналоговая величина
<b>Q</b>	INT	= value1 if SEL = 0 = value2 if SEL = 1 ... = value8 if selector = 7 = 0 для всех других значений селектора

Описание:

Мультиплексор 8 входов: выбирает одно из восьми целых чисел.

(\*FBD пример блока "MUX8"\*)



(\* ST Эквивалент: \*)

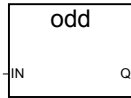
range := MUX8 (choice, 1, 5, 10, 50, 100, 500, 1000, 5000);

(\* выбирает из 8 predeterminedенных значений, например, если выбрана 3, значение будет 50 \*)

(\* IL Эквивалент: \*)

LD choice  
MUX8 1,5,10,50,100,500,1000,5000  
ST range

## ODD



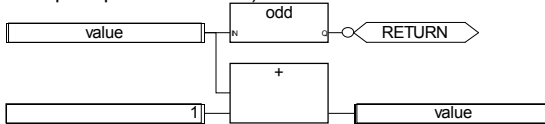
Аргументы:

<b>IN</b>	INT	любая целая знаковая аналоговая величина
<b>Q</b>	BOO	TRUE если входная величина нечетная FALSE если входная величина четная

Описание:

Проверяет целую величину на четность: результат - четный или нечетный.

(\*FBD пример блока "ODD"\*)



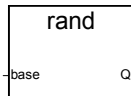
(\* ST Эквивалент: \*)

```
If Not (ODD (value)) Then Return; End_if;  
value := value + 1;  
(*делает значение четным*)
```

(\* IL Эквивалент: \*)

```
LD value  
ODD  
RETNC  
LD value  
ADD 1  
ST value
```

## RAND



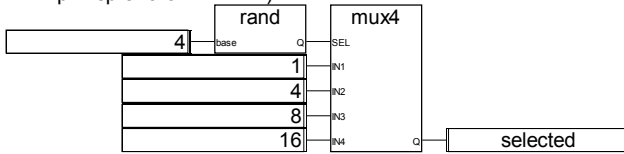
Аргументы:

<b>base</b>	INT	определяет допустимое множество чисел
<b>Q</b>	INT	случайная величина в диапазоне [0..base - 1]

Описание:

Дает случайную целую величину в заданном диапазоне.

(\*FBD пример блока "RAND"\*)



(\* ST Эквивалент: \*)

```
selected := MUX4 ( RAND (4), 1, 4, 8, 16 );
```

(\*

случайный выбор одного из 4 predetermined значений

RAND выдает значение в интервале [0..3],

так 'selected' выходящий из MUX4, получит случайное значение

1 если 0 выходит из RAND,

или 4 если 1 выходит из RAND,

или 8 если 2 выходит из RAND,

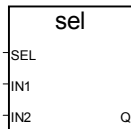
или 16 если 3 выходит из RAND,

\*)

(\* IL Эквивалент: \*)

```
LD      4
RAND
MUX4    1,4,8,16
ST      selected
```

## SEL



Аргументы

<b>SEL</b>	BOO	определяет выбранную величину
<b>IN1, IN2</b>	INT	любая целая аналоговая величина
<b>Q</b>	INT	= value1 если SEL = FALSE = value2 если SEL = TRUE

Описание:

Двоичный селектор: выбирает значение из двух целых чисел.

(\*FBD пример блока "SEL"\*)



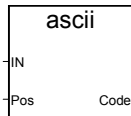
\_ (\* ST Эквивалент: \*)

ProCmnd := SEL (AutoMode, ManuCmnd, InpCmnd);

(\* выбор команды \*)

	<b>IN</b>	MSG	любая непустая строка
LD	AutoMode		
SEL	ManuCmnd, InpCmnd		
ST	ProCmnd		

## ASCII



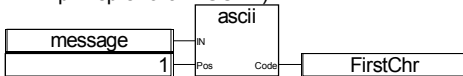
Аргументы:

<b>IN</b>	MSG	любая непустая строка
<b>Pos</b>	INT	позиция выбранного символа в диапазоне [1..len] (len - длина сообщения IN)
<b>Code</b>	INT	код выбранного символа (в диапазоне [0..255]) возвращает 0 если Pos вне строки

Описание:

Дает ASCII код символа в строке.

(\*FBD пример блока "ASCII"\*)



(\* ST Эквивалент: \*)

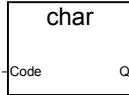
FirstChr := ASCII (message, 1);

(\* FirstChr - это ASCII код первого символа строки \*)

(\* IL Эквивалент: \*)

LD	message
ASCII	1
ST	FirstChr

## CHAR



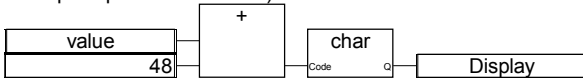
Аргументы:

<b>Code</b>	INT	код в диапазоне [0..255]
<b>Q</b>	MSG	строка из одного символа символ имеет ASCII код заданный на входе (используется ASCII код по модулю 256)

Описание:

Дает строку из одного символа с заданным ASCII кодом.

(\*FBD пример блока "CHAR"\*)



(\* ST Эквивалент: \*)

Display := CHAR ( value + 48 );

(\* значение в диапазоне [0..9] \*)

(\* 48 - это ascii код '0' \*)

(\* result - строка из одного символа от '0' до '9' \*)

(\* IL Эквивалент: \*)

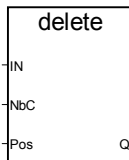
LD value

ADD 48

CHAR

ST Display

## DELETE



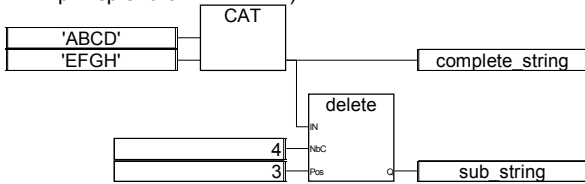
Аргументы:

<b>IN</b>	MSG	любая непустая строка
<b>NbC</b>	INT	количество символов, которые нужно удалить
<b>Pos</b>	INT	позиция первого символа для удаления (позиция первого символа строки - 1)
<b>Q</b>	MSG	измененная строка пустая строка если Pos < 1 первоначальная строка если Pos > длина IN первоначальная строка если NbC <= 0

Описание:

Удаляет часть строки.

(\*FBD пример блока "DELETE"\*)



(\* ST Эквивалент: \*)

complete\_string := 'ABCD' + 'EFGH'; (\* полная строка - это 'ABCDEFGH' \*)  
 sub\_string := DELETE (complete\_string, 4, 3); (\* sub\_string is 'ABGH' \*)

(\* IL Эквивалент: \*)

```
LD      'ABCD'
ADD     'EFGH'
ST      complete_string
DELETE  4,3
ST      sub_string
```

## FIND



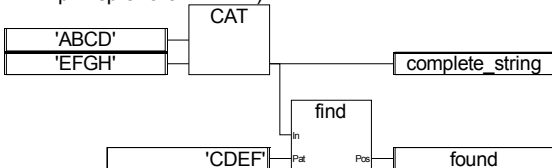
Аргументы:

<b>IN</b>	MSG	любая строка
<b>Pat</b>	MSG	любая непустая строка
<b>Pos</b>	INT	=0 если подстрока Pos не найдена =позиция первого символа первого вхождения подстроки Pat (первая позиция - 1)      эта функция отличает заглавные буквы от прописных

Описание:

Находит подстроку в строке. Возвращает положение подстроки в строке.

(\*FBD пример блока "FIND"\*)



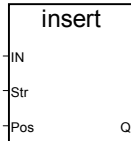
(\* ST Эквивалент: \*)

complete\_string := 'ABCD' + 'EFGH'; (\* полная строк - это 'ABCDEFGH' \*)  
 found := FIND (complete\_string, 'CDEF'); (\* найдено 3 \*)

(\* IL Эквивалент: \*)

```
LD      'ABCD'  
ADD     'EFGH'  
ST      complete_string  
FIND    'CDEF'  
ST      found
```

## INSERT



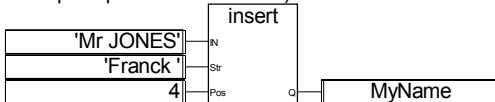
Аргументы:

<b>IN</b>	MSG	начальная строка
<b>Str</b>	MSG	строка которую нужно вставить
<b>Pos</b>	INT	позиция вставки вставка делается перед позицией (первая позиция - 1)
<b>Q</b>	MSG	измененная строка пустая строка если Pos < 0 соединение строк если Pos > длина IN

Описание:

Вставляет подстроку в строку начиная с данной позиции.

(\*FBD пример блока "INSERT"\*)



(\* ST Эквивалент: \*)

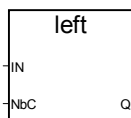
```
MyName := INSERT ('Mr JONES', 'Frank ', 4);
```

(\* MyName - это 'Mr Frank JONES' \*)

(\* IL Эквивалент: \*)

```
LD      'Mr JONES'  
INSERT  'Frank ',4  
ST      MyName
```

## LEFT





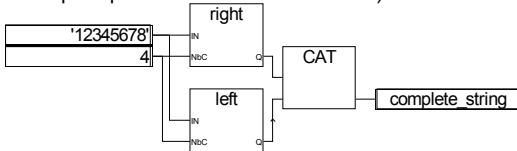
Аргументы:

<b>IN</b>	MSG	любая непустая строка
<b>NbC</b>	INT	Количество символов, которые нужно изъять не может быть больше строки IN
<b>Q</b>	MSG	левая часть строки IN (ее длина = NbC) пустая строка если NbC < 0 вся строка IN если NbC > длина IN

Описание:

Берет левую часть строки. Количество символов задано.

(\*FBD пример блоков "LEFT" и "RIGHT"\*)



(\* ST Эквивалент: \*)

complete\_string := RIGHT ('12345678', 4) + LEFT ('12345678', 4);

(\* полная строка - это '56781234'

значение выходящее из RIGHT - это '5678'

значение выходящее из LEFT - это '1234'

\*)

(\* IL Эквивалент: Сначала вызывается LEFT \*)

LD '12345678'

LEFT 4

ST sub\_string (\* промежуточный результат \*)

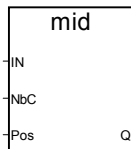
LD '12345678'

RIGHT 4

ADD sub\_string

ST complete\_string

## MID



Аргументы:

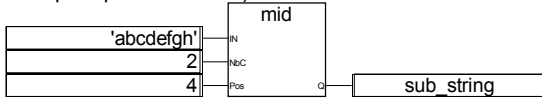
<b>IN</b>	MSG	любая непустая строка
<b>NbC</b>	INT	Количество символов, которые нужно изъять не может быть больше длины строки IN
<b>Pos</b>	INT	позиция подстроки Pos должен указывать на первый символ подстроки (первая правильная позиция 1)

## Q

MSG средняя часть строки IN (ее длина = NbC)  
пустая строка если параметры неправильные

Описание:

Берет часть строки. Количество символов и позиция первого символа заданы.  
(\*FBD пример блока "MID"\*)



(\* ST Эквивалент: \*)

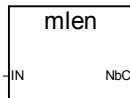
```
sub_string := MID ('abcdefgh', 2, 4);
```

(\* подстрока - 'de' \*)

(\* IL Эквивалент: \*)

```
LD      'abcdefgh'  
MID     2,4  
ST      sub_string
```

## MLEN



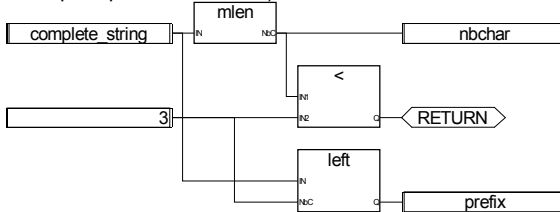
Аргументы:

<b>IN</b>	MSG	любая непустая строка
<b>NbC</b>	INT	Количество символов в строке IN

Описание:

Вычисляет длину строки.

(\*FBD пример блока "MLEN"\*)



(\* ST Эквивалент: \*)

```
nbchar := MLEN (complete_string);
```

```
If (nbchar < 3) Then Return; End_if;
```

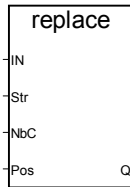
```
prefix := LEFT (complete_string, 3);
```

(\*эта программа извлекает 3 символа из левой части строки кладет результат в строковую переменную prefix никаких действий не выполняется если длина строки менее чем 3 символа\*)

(\* IL Эквивалент: \*)

LD complete\_string  
MLEN  
ST nbchar  
LT 3  
RETC  
LD complete\_string  
LEFT 3  
ST prefix

## REPLACE



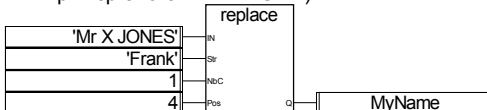
Аргументы:

<b>IN</b>	MSG	любая строка
<b>Str</b>	MSG	строка, которую нужно вставить
<b>NbC</b>	INT	Количество символов, которые должны быть удалены
<b>Pos</b>	INT	Позиция первого измененного символа (первая правильная позиция 1)
<b>Q</b>	MSG	измененная строка: -NbC символов удаляются начиная с позиции Pos -затем строка Str вводится начиная с этой позиции возвращает пустую строку если Pos <= 0 возвращает соединение строк (IN+Str) если Pos больше, чем длина строки IN возвращает начальную строку IN если NbC <= 0

Описание:

Заменяет часть строки новым набором символов.

(\*FBD пример блока "REPLACE"\*)



(\* ST Эквивалент: \*)

MyName := REPLACE ('Mr X JONES', 'Frank', 1, 4);

(\* MyName - это 'Mr Frank JONES' \*)

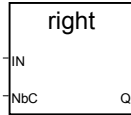
(\* IL Эквивалент: \*)

LD 'Mr X JONES'  
REPLACE 'Frank',1,4

ST

MyName

## RIGHT



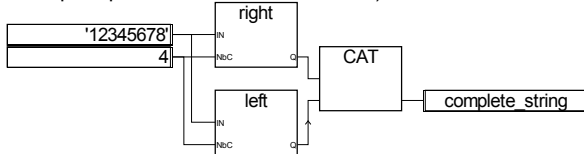
Аргументы:

<b>IN</b>	MSG	любая непустая строка
<b>NbC</b>	INT	не может быть больше, чем длина строки IN
<b>Q</b>	MSG	правая часть строки IN (ее длина = NbC) пустая строка если NbC < 0 вся строка IN если NbC > длина IN

Описание:

Берет правую часть строки. Количество символов задано.

(\*FBD пример блоков "LEFT" и "RIGHT"\*)



(\* ST Эквивалент: \*)

```
complete_string := RIGHT ('12345678', 4) + LEFT ('12345678', 4);
```

(\* полная строка - это '56781234'

значение выходящее из RIGHT - это '5678'

значение выходящее из LEFT - это '1234'

\*)

(\* IL Эквивалент: Сначала вызывается LEFT \*)

```
LD '12345678'
```

```
LEFT 4
```

```
ST sub_string (* промежуточный результат *)
```

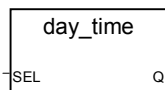
```
LD '12345678'
```

```
RIGHT 4
```

```
ADD sub_string
```

```
ST complete_string
```

## DAY\_TIME



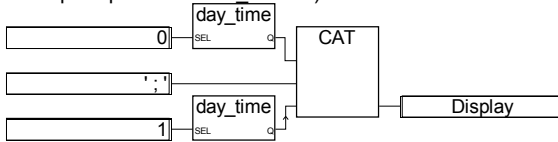
Аргументы:

<b>SEL</b>	INT	выбор выхода 0=текущая дата 1=текущее время 2=день недели
<b>Q</b>	MSG	время/дата, выраженные символьной строкой 'YYY/MMM/DDD' если SEL = 0 'HH:MM:SS' если SEL = 1 имя дня если SEL = 2 (например: 'Monday')

Описание:

Возвращает дату или время дня, выраженные символьной строкой.

(\*FBD пример блока "DAY\_TIME"\*)



(\* ST Эквивалент: \*)

Display := Day\_Time (0) + ' ; ' + Day\_Time (1);

(\*Формат текста: 'YYYY/MM/DD ; HH:MM:SS' \*)

(\* IL Эквивалент: Сначала вызывается day\_time(1) \*)

LD 1

DAY\_TIME

ST hour\_str (\* промежуточный результат \*)

LD 0

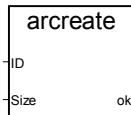
DAY\_TIME

ADD ' ; '

ADD hour\_str

ST Display

## ARCREATE



Аргументы

<b>ID</b>	INT	идентификатор массива (должен быть в пределах [0..15])
<b>Size</b>	INT	количество элементов в массиве
<b>Q</b>	INT	результат исполнения 1 = если массив был успешно создан 2 = неправильный идентификатор массива или массив уже создан 3 = неправильный размер

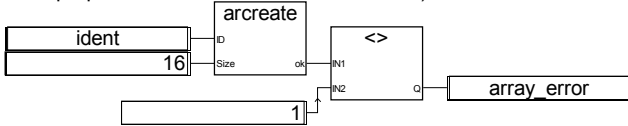
4 = не хватило памяти

Описание:

Создание массива целых.

Предупреждение: В приложении может быть не более 16 массивов. Массив содержит целые аналоговые значения. Если размер массива близок к размеру имеющейся свободной памяти, то эта функция может вызвать системную ошибку при захвате памяти.

(\*FBD программа, создающая массив целых\*)



(\* ST Эквивалент: \*)

```
array_error := (ARCREATE (ident, 16) <> 1));
```

(\* IL Эквивалент: \*)

```
LD      ident
ARCREATE 16
NE      1
ST      array_error
```

## ARREAD



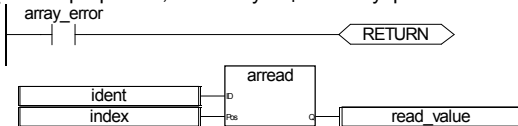
Аргументы:

<b>ID</b>	INT	идентификатор массива (должен быть в пределах [0..15])
<b>Pos</b>	INT	позиция элемента в массиве (должна быть в пределах [0..size-1])
<b>value</b>	INT	значение считанного элемента 0 = если аргумент неправильный

Описание:

Читает элемент массива целых.

(\*FBD программа, использующая блок управления массивом\*)



(\* ST Эквивалент: \*)

```
If (array_error) Then Return; End_if;
```

```
read_value := ARREAD (ident, index);
(* array_error приходит от ARCREATE call *)
```

```
(* IL Эквивалент: *)
LD          array_error
RETC
LD          ident
ARREAD     index
ST          read_value
```

## ARWRITE



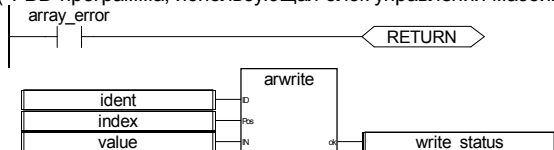
Аргументы:

<b>ID</b>	INT	идентификатор массива (должен быть в пределах [0..15])
<b>Pos</b>	INT	позиция элемента в массиве (должна быть в пределах [0..size-1])
<b>IN</b>	INT	новое значение элемента
<b>ok</b>	INT	результат исполнения 1 = запись была успешно выполнена 2 = неправильный идентификатор массива 3 = неправильный индекс

Описание:

Запоминает (записывает) значение в массив целых.

(\*FBD программа, использующая блок управления массивом\*)



(\* ST Эквивалент: \*)

```
If (array_error) Then Return; End_if;
write_status := ARWRITE (Ident, Index, value);
(* array_error приходит от ARCREATE *)
```

(\* IL Эквивалент: \*)

```
LD          array_error
RETC
LD          ident
ARWRITE     index,value
ST          write_status
```

## F\_ROPEN



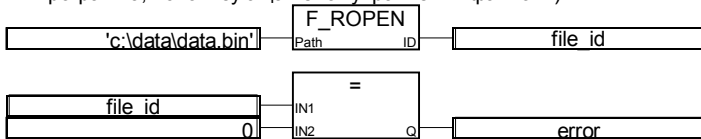
Аргументы:

<b>Path</b>	MSG	имя файла может определять путь доступа к файлу с использованием символов / или \ для определения директории. Для переносимости приложений / или \ эквивалентны
<b>ID</b>	INT	номер файла 0 если была ошибка: файл не существует

Описание:

Открывает двоичный файл для чтения. Должен использоваться с FX\_READ и FX\_CLOSE. Эта функция не включена в симулятор ISaGRAF.

(\*FBD программа, использующая блок управления файлом\*)



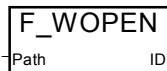
(\* ST Эквивалент: \*)

```
file_id := F_ROPEN('c:\data\data.bin');  
error := (file_id=0);
```

(\* IL Эквивалент: \*)

```
LD      'c:\data\data.bin'  
F_ROPEN  
ST      file_id  
EQ      0  
ST      error
```

## F\_WOPEN



Аргументы:

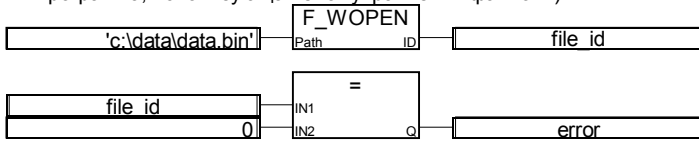
<b>Path</b>	MSG	имя файла может определять путь доступа к файлу с использованием символов / или \ для определения директории. Для переносимости приложений / или \ эквивалентны.
<b>ID</b>	INT	номер файла 0 если была ошибка: файл не существует



Описание:

Открывает двоичный файл для чтения. Должен использоваться с FX\_READ и FX\_CLOSE. Эта функция не включена в симулятор ISaGRAF.

(\*FBD программа, использующая блок управления файлом\*)



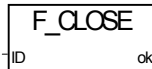
(\* ST Эквивалент: \*)

```
file_id := F_WOPEN('c:\data\data.bin');
error := (file_id=0);
```

(\* IL Эквивалент: \*)

```
LD      'c:\data\data.bin'
F_WOPEN
ST      file_id
EQ      0
ST      error
```

## F\_CLOSE



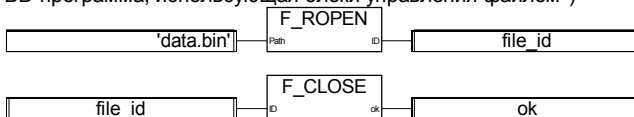
Аргументы:

<b>ID</b>	INT	номер файла: возвращенный функциями F_ROPEN или F_WOPEN
<b>ok</b>	BOO	статус TRUE если файл закрыт успешно FALSE если была ошибка

Описание:

Закрывает двоичный файл, открытый функциями F\_ROPEN или F\_WOPEN. Эта функция не включена в симулятор ISaGRAF.

(\*FBD программа, использующая блоки управления файлом\*)



```
(* ST Эквивалент: *)
file_id := F_ROPEN('data.bin');
ok := F_CLOSE(file_id);
```

```
(* IL Эквивалент: *)
LD      'data.bin'
F_ROPEN
ST      file_id
F_CLOSE      (* идентификатор файла уже в текущем результате *)
ST      ok
```

## F\_EOF



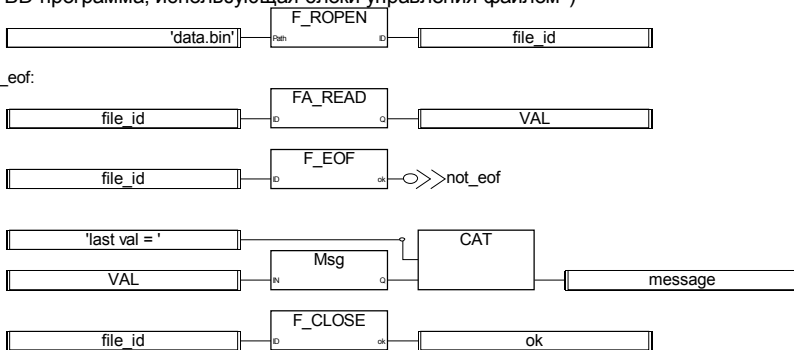
Аргументы:

<b>ID</b>	INT	номер файла: возвращенный функциями F_ROPEN или F_WOPEN
<b>ok</b>	BOO	индикатор конца файла TRUE если конец файл был достигнут при последнем чтении или записи FM_READ последнее сообщение прочитанное из файла может быть неверным, если последний символ - не символ окончания строки

Описание:

Проверяет достигнут ли конец файла. Эта функция не включена в симулятор ISaGRAF.

(\*FBD программа, использующая блоки управления файлом\*)



```
(* ST Эквивалент: *)
file_id := F_ROPEN('data.bin');
WHILE not(F_EOF(file_id))
```

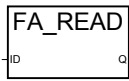
```

        VAL := FA_READ(file_id);
END_WHILE;
MESSAGE := 'last val = ' + msg(VAL);
ok := F_CLOSE(file_id);

(* IL Эквивалент: *)
        LD          'data.bin'
        F_ROPEN
        ST          file_id
        LD          file_id
        F_EOF
        JMPNC      END_OF_FILE
NOT_EOF:  LD          file_id
        FA_READ
        ST          VAL
        LD          file_id
        F_EOF
        JMPNC      NOT_EOF (* если не eof, то читать *)
END_OF_FILE: LD      VAL
        MSG
        ST          val_msg          (* превращение VAL в строку *)
        LD          'last val = '
        ADD         val_msg
        ST          MESSAGE
        LD          file_id
        F_CLOSE
        ST          ok

```

**FA\_READ**

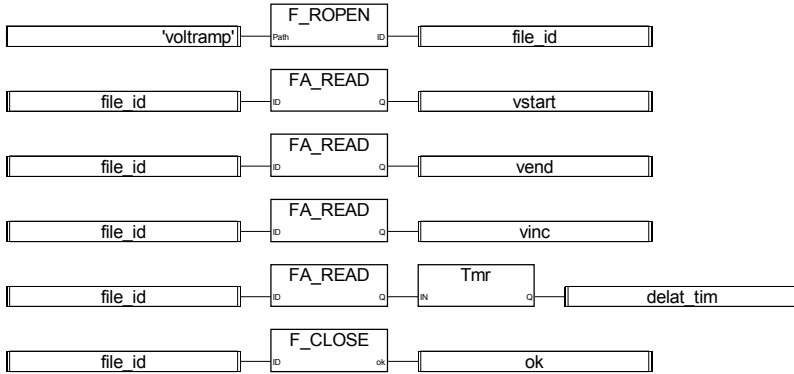


Аргументы:

<b>ID</b>	INT	номер файла: возвращенный функцией F_ROPEN
<b>Q</b>	INT	целая аналоговая величина считанная из файла

Описание:  
 Читает аналоговые переменные из двоичного файла. Должна использоваться с F\_ROPEN и F\_CLOSE. Эта функция осуществляет последовательный доступ к файлу, начиная с предыдущей позиции. Первый вызов после F\_ROPEN читает первые 4 байта файла, каждый вызов сдвигает указатель чтения. Для проверки достигнут ли конец файла используйте F\_EOF. Эта функция не включена в симулятор ISaGRAF.

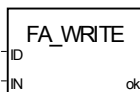
(\*FBD программа, использующая блоки управления файлом\*)



```
(* ST Эквивалент: *)
file_id := F_ROPEN('voltramp.bin');
vstart := FA_READ(file_id);
vend := FA_READ(file_id);
vinc := FA_READ(file_id);
delta_tim := tmr(FA_READ(file_id));
ok := F_CLOSE(file_id);
```

```
(* IL Эквивалент: *)
LD      'voltramp.bin'
F_ROPEN
ST      file_id
FA_READ      (* читать vstart *)
ST      vstart
LD      file_id
FA_READ      (* читать vend *)
ST      vend
LD      file_id
FA_READ      (* читать vinc *)
ST      vinc
LD      file_id
FA_READ      (* читать delta_tim *)
TMR      (* превращение в timer *)
ST      delta_tim
LD      file_id
F_CLOSE
ST      ok
```

## FA\_WRITE



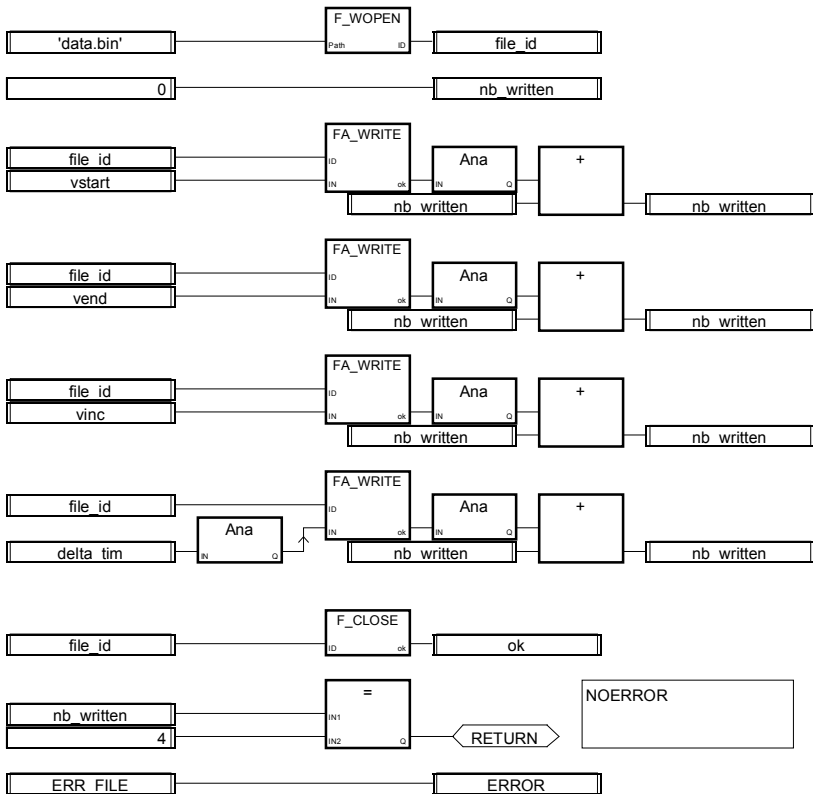
Аргументы:

<b>ID</b>	INT	номер файла: возвращенный функцией F_WOPEN
<b>IN</b>	INT	целая аналоговая величина, которую нужно записать в файла.
<b>Q</b>	BOO	статус выполнения: TRUE если ок

Описание:

Записывает аналоговые переменные в двоичный файл. Эта функция осуществляет последовательный доступ к файлу, начиная с предыдущей позиции. Первый вызов после F\_WOPEN записывает первые 4 байта файла, каждый вызов сдвигает указатель чтения. Для проверки достигнут ли конец файла используйте F\_EOF. Эта функция не включена в симулятор ISaGRAF.

(\*FBD программа\*)



(\* ST Эквивалент: \*)

```
file_id := F_WOPEN('voltramp.bin');
nb_written := 0;
```

```

nb_written := nb_written + ana(FA_WRITE(file_id,vstart));
nb_written := nb_written + ana(FA_WRITE(file_id,vend));
nb_written := nb_written + ana(FA_WRITE(file_id,vinc));
nb_written := nb_written + ana(FA_WRITE(file_id,ana(delta_tim)));
ok := F_CLOSE(file_id);
IF ( nb_written <> 4) THEN
    ERROR := ERR_FILE;
END_IF;

```

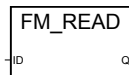
(\* IL Эквивалент: \*)

```

LD      'voltramp.bin'
F_ROPEN
ST      file_id
LD      0
ST      nb_written
LD      file_id      (* писать vstart *)
FA_WRITE vstart
ANA
ADD     nb_written
ST      nb_written
LD      file_id      (* писать vend *)
FA_WRITE vend
ANA
ADD     nb_written
ST      nb_written
LD      file_id      (*писать vinc *)
FA_WRITE vinc
ANA
ADD     nb_written
LD      nb_written      (* писать delta_tim *)
ANA      (* превратить в 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      (* возврат если равно 4 *)
LD      ERR_FILE (* иначе ошибка *)
ST      ERROR

```

## FM\_READ



Аргументы:

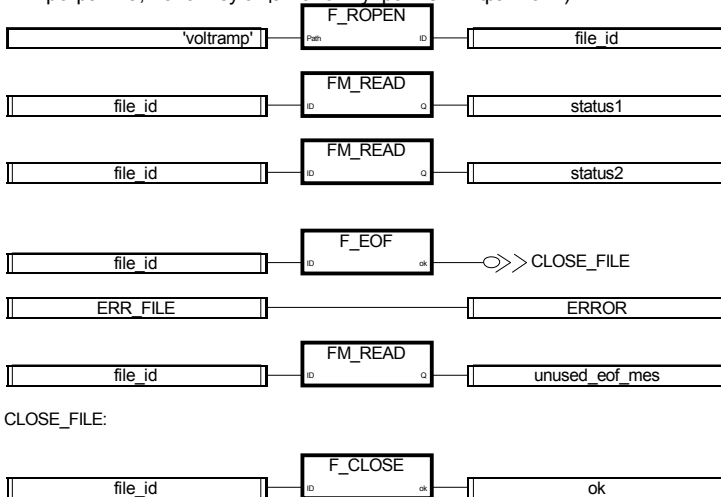
<b>ID</b>	INT	номер файла: возвращенный функцией F_ROPEN
<b>Q</b>	MSG	сообщение считанное из файла

Описание:

Читает строковые переменные из двоичного файла. Должна использоваться с F\_ROPEN и F\_CLOSE. Эта функция осуществляет последовательный доступ к файлу, начиная с предыдущей позиции. Первый вызов после F\_ROPEN читает первую строку файла, каждый вызов сдвигает указатель чтения. Строка заканчивается нулем (0), символом конца строки ('\n') или возвратом ('\r');

Для проверки достигнут ли конец файла используйте F\_EOF. Эта функция не включена в симулятор ISaGRAF.

(\*FBD программа, использующая блоки управления файлом\*)



(\* ST Эквивалент: \*)

```
file_id := F_ROPEN('voltramp.bin');
status1 := FM_READ(file_id);
status2 := FM_READ(file_id);
IF (F_EOF(file_id)) THEN
    ERROR := ERR_FILE;
    unused_eof_mes := FM_READ(file_id);
END_IF;
ok := F_CLOSE(file_id);
```

(\* IL Эквивалент: \*)

```
LD      'voltramp.bin'
F_ROPEN
ST      file_id
```

```

FM_READ      (* читать status1 *)
ST          status1
LD          file_id
FM_READ      (* читать status2 *)
ST          status2
LD          file_id
F_EOF
JMPNC      CLOSE_FILE      (* если конец файла, то не делать прыжок *)

*)

LD          ERR_FILE
ST          ERROR
LD          file_id
FM_READ      (* читать unused_eof_mes *)
ST          unused_eof_mes
CLOSE_FILE  LD          file_id
F_CLOSE
ST          ok

```

## FM\_WRITE



Аргументы:

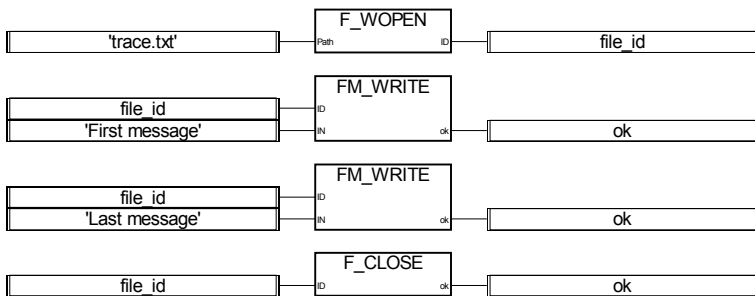
<b>ID</b>	INT номер файла: возвращенный функцией F_ROPEN
<b>IN</b>	MSG сообщение, которое надо записать в файл
<b>ok</b>	BOO статус выполнения TRUE если успешно

Описание:

Записывает строковые переменные в двоичный файл. Должна использоваться с F\_WOPEN и F\_CLOSE. Эта функция осуществляет последовательный доступ к файлу, начиная с предыдущей позиции. Первый вызов после F\_WOPEN записывает первую строку файла, каждый вызов сдвигает указатель записи. Эта функция не включена в симулятор ISaGRAF.

(\*FBD программа, использующая блоки управления файлом\*)





(\* ST Эквивалент: \*)

```

file_id := F_WOPEN('trace.txt');
ok := FM_WRITE(file_id,'First message');
ok := FM_WRITE(file_id,'Last message');
ok := F_CLOSE(file_id);
  
```

(\* IL Эквивалент: \*)

```

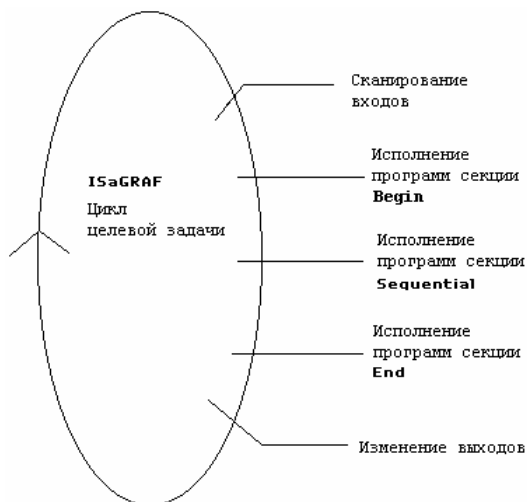
LD      'trace.txt'
F_WOPEN
ST      file_id
FM_WRITE 'First message'      (*писать первое сообщение *)
ST      ok
LD      file_id
FM_WRITE 'Last message'      (* писать второе сообщение *)
ST      ok
LD      file_id
F_CLOSE
ST      ok
  
```



# **С. Руководство пользователя целевой задачи**

## С.1 Введение

Целевая задача ISaGRAF работает на контроллере в соответствии с хорошо известной схемой:



Цикл целевой задачи<sup>1</sup> состоит из сканирования физических входов, обработки данных в соответствии с прикладной программой и изменения физических выходов.

- Первая часть этой главы объясняет, как начать работать с целевой задачей ISaGRAF для DOS, OS-9, VxWorks и NT. Во-первых, вы найдете, как запустить целевую задачу ISaGRAF для каждой из этих платформ. После этого, вы получите информацию о таких специфических особенностях как: запуск целевой задачи при включении питания, обработке ошибок, поведении в целом и т.д.
- Вторая часть имеет дело с методами реализации пользовательских “С” функций, функциональных блоков и функций преобразования для улучшения целевой задачи ISaGRAF.
- В третьей части содержится информация о Modbus и реализации ISaGRAF. В ней рассматриваются форматы кадров для различных кодов функций.
- В четвертой части рассматриваются средства преодоления отказов питания и рестарт целевой задачи ISaGRAF.

<sup>1</sup> Это руководство подразумевает, что пользователь знаком с целевой задачей ISaGRAF

## С.2 Установка

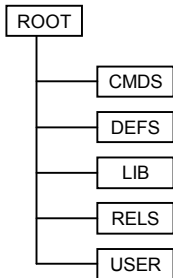
Для установки требуется около 1 мегабайта памяти на диске.

Файл `install.bat`, находящийся на установочном диске установит все необходимые файлы для указанной платформы на ваш PC.

Пример: `a:\install a: c:\path`

устанавливает файлы с `a:` на `c:` в каталог `path`.

Создается следующая структура каталогов:



Каталог `ROOT` содержит некоторые инструментальные средства и файлы `readme`

Каталог `CMDS` содержит исполняемые файлы

Каталог `DEFS` содержит файлы заголовков

Каталог `LIB` содержит библиотеки

Каталог `RELS` содержит объектные файлы

Каталог `USER` содержит 'C' файлы пользователя, функциональные блоки и функции преобразований (исходники и файлы заголовков)

После этого можно начинать работу.

## С.3 Работа с целевой задачей ISaGRAF DOS

### С.3.1 Работа ISaGRAF: ISA.EXE

В реализации под MS-DOS целевая задача представляет собой отдельную программу ISA.EXE. При необходимости можно получить подсказку с помощью команды isa -? из каталога CMDS.

Такая конфигурация критична по ресурсам, например, не рекомендуется перегружать коммуникации для того, чтобы гарантировать хорошую производительность. Целевая задача не мешает работе драйверов, имеющих обработчики прерывания.

#### ⇒ **Конфигурирование коммуникаций: -t ключ**

Целевая задача ISaGRAF использует последовательный порт для связи с отладчиком. Название порта определяется при помощи ключа -t. Коммуникационный интерфейс учитывает особенности различных машин, поэтому можно использовать порты COM1, COM2 или COM3 в зависимости от версии BIOS.

**Нет значения по умолчанию:** Если эта опция не использована, то связь с целевой задачей невозможна. В таком случае может быть выдано сообщение об ошибке номер 7.

В DOS связь по ETHERNET не предусмотрена. Такая реализация может быть обеспечена вашим поставщиком.

Параметры последовательного порта должны быть установлены до запуска ISaGRAF. При использовании отладчика системы разработки убедитесь, что параметры порта соответствуют тем, которые установлены в целевой задаче (см. руководство пользователя): Управляющие программы.

#### Пример:

устанавливает следующие значения параметров порта:

```
скорость 9600 бит/с  
нет контроля четности  
данные 8 бит  
1 стоп-бит
```

Заметим, что в некоторых версиях BIOS значение скорости 19200, принятое по умолчанию в системе разработки, не разрешено.

Для установки параметров порта CJ предоставляет утилиту ISAMOD.EXE. Вызов

```
ISAMOD COM1
```

эквивалентен

```
MODE COM1:9600,N,8,1
```

#### ⇒ **Номер подчиненного: -s ключ**

Эта опция определяет номер подчиненного целевой задачи. Он может принимать значения от 1 до 255, исключая 13 (\$0D). Этот номер используется в протоколе связи. Он нужен для того, чтобы отличать целевые задачи друг от друга, когда запущено несколько задач. Когда используется отладчик, убедитесь, что его подчиненный параметр соответствует номеру целевой задачи.

**По умолчанию** номер подчиненного равен 1.

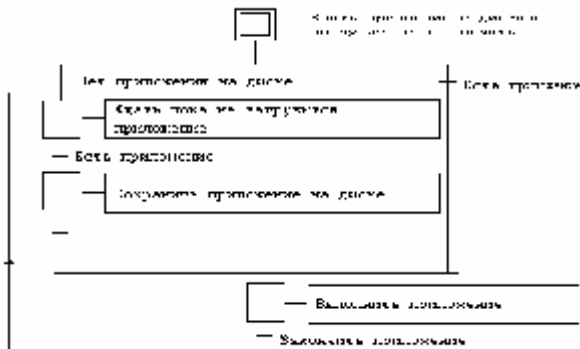
## Примеры:

- isamod COM1** устанавливает на COM1: 19200 бод, без четности, данные 8 бит, 1 стоп бит
- isa -t=COM1** запуск целевой задачи ISaGRAF как подчиненный 1 с портом связи COM1
- isa -s=3 -t=COM1** запуск целевой задачи ISaGRAF как подчиненный 3 с портом связи COM1

### С.3.2 Специфические особенности

#### Запуск ISaGRAF

В начале работы ISaGRAF выполняется следующий алгоритм:



#### • Определения

Код приложения это двоичный код, который генерируется и загружается подсистемой разработки и затем выполняется на целевой машине. Он может быть дополнен таблицей символов.

Таблица символов - это ASCII база данных, которая генерируется и загружается подсистемой разработки. Эта таблица обеспечивает связь символьных объектов и внутренних объектов целевой задачи. Она требуется в целевой задаче только в случаях специфического управления символами. Подробнее о таблице символов можно узнать в руководстве пользователя: Продвинутые средства программирования.

Когда приложение загружается отладчиком подсистемы разработки в целевую машину, код приложения сохраняется на диске в файле под именем

**ISAx1** резервная копия файла кода приложения ISaGRAF (x - номер подчиненного)

Если, кроме того загружена таблица символов, она тоже сохраняется на диске в файле под именем

**ISAx6** резервная копия файла таблицы символов приложения (x - номер подчиненного)

Когда ISaGRAF запускается, целевая задача ищет код приложения и таблицу символов в текущем каталоге и загружает их в память. Если файл таблицы символов не найден, то целевая задача исполняет код приложения без таблицы символов. Если в памяти нет кода приложения, то целевая задача ждет, когда приложение будет загружено.

Для того чтобы запустить приложение при включении машины без использования отладчика скопируйте эти файлы на диск целевой машины в текущий каталог целевой задачи. В бездискковой конфигурации можно использовать виртуальный диск.

Если система разработки ISaGRAF установлена в стандартном каталоге \ISAWIN, то: файл кода приложения проекта MYPROJ будет называться

```
ISAWIN\APL\MYPROJ\appli.x8m
```

файл таблицы символов приложения проекта MYPROJ будет называться

```
ISAWIN\APL\MYPROJ\appli.tst
```

#### Пример:

Если из каталога, где находится isa.exe, выдать команду

```
copy ISAWIN\APL\MYPROJ\appli.x8m isa11
```

то isa.exe найдет и выполнит приложение 'myproj'.

Все эти команды можно поместить в командный файл и запускать его из меню Инструменты системы разработки.

## == **Обработка ошибок и выдача сообщений**

Целевая задача ISaGRAF включает в себя обработчик ошибок. Список ошибок приводится в приложении.

Обработка ошибок происходит по следующим правилам:

- Ошибка состоит из номера и аргумента, посылаемых обработчику ошибок ISaGRAF.
- Если в системе разработки установлен флаг обнаружения ошибок, то ошибка обрабатывается. Если нет - информация теряется и обработка заканчивается.

Если обрабатывается:

- Номер (десятичное число) и аргумент (шестнадцатеричное число) выводятся в stdout.
- Номер ошибки и аргумент помещаются в кольцевой буфер FIFO, а позднее переносятся оттуда. Размер буфера определяется в опциях Make системы разработки. Когда буфер полон, сообщение о новой ошибке вытесняет самое старое сообщение.
- Ошибки извлекаются из буфера либо отладчиком, либо приложением, используя вызов SYSTEM (см. Руководство пользователя).

Когда отладчик фиксирует ошибку, сообщение о ней появляется в окне отображения ошибок. В зависимости от ситуации (работает приложение или нет) отладчик отображает имя объекта (переменной или программы) откуда пришла ошибка или аргумент ошибки (десятичное число) в квадратных скобках [x], который имеет различное значение для разных ошибок.

Сообщение об ошибке и ее значение по умолчанию выдаются в stdout. Если отображение в stdout нежелательно, можно использовать команду перенаправления следующего вида

```
isa -t=COM1 -s=1 >NUL
```

## == **Системные часы**

Поскольку ISaGRAF должен работать на любой системе, синхронизация цикла и обновление значений таймера происходит квантами величиной один стандартный тик (около 55 миллисекунд).



Следовательно, разрешение таймера не может быть лучше 55 мсек. По этой же причине при попытке остановить продолжительность цикла менее 55 мсек. (но отличную от 0) будет выдаваться ошибка переполнения продолжительности цикла (error 66).

Невозможность модифицировать системный тик имеет и положительную сторону: исполнение ISaGRAF не может помешать никаким системным приложениям, а также функциям С или функциональным блокам, интегрированным в приложение.

Обратитесь к поставщику, если требуется специальная реализация, требующая лучшего разрешения.

## ⇒ **Выход из системы**

При тестировании приложения в непроизводственных условиях на настольном ПК у пользователя может возникнуть необходимость остановить ISaGRAF: это делается одновременным нажатием трех клавиш (чтобы исключить случайную остановку)

### **shift + ctrl + alt**

Разумеется, если приложение не должно останавливаться по нажатию клавиш, необходимо предусмотреть какие-то средства, чтобы запретить такие комбинации.

Один из опасных побочных эффектов быстрого выхода заключается в том, что при этом не отключается интерфейс с платами ввода/вывода. Поэтому предусмотрены аккуратные способы остановки целевой задачи ISaGRAF:

остановка приложения из отладчика (при этом интерфейс с платами в/в закрывается)

остановки целевой задачи ISaGRAF с клавиатуры

## ⇒ **Размер приложения**

Поскольку целевая задача ISaGRAF под MSDOS предназначена для работы в реальном режиме, размер структуры данных не может превышать 64К. Поэтому код приложения, загружаемый системой разработки, не должен превышать этот предел. Вся доступная память ограничена стандартными 640 килобайтами.

Если требования по памяти Вашего приложения превышают эти пределы - обратитесь к поставщику для специальной реализации.

## C.4 Работа с целевой задачей ISaGRAF OS9

Для того чтобы начать работу с целевой задачей ISaGRAF, прежде всего, нужно переписать программы на целевую машину OS-9. Затем, для того чтобы начать работать, можно просто запустить команду помощи:

```
isa -?  
isaker -?  
isatst -?  
Isanet -?
```

### C.4.1 Работа ISaGRAF в однозадачном режиме: isa

ISaGRAF может работать как одна задача. В таком случае не рекомендуется перегружать коммуникации для того, чтобы гарантировать хорошую производительность. В многозадачной системе OS-9 различные целевые задачи ISaGRAF могут быть запущены на одном и том же процессоре, если они имеют различные номера подчиненных и последовательные порты.

Однозадачная реализация, в основном ориентирована на разработку первого прототипа системы при портировании на новую платформу. Многозадачная реализация более предпочтительна.

Задача ISaGRAF не нарушает работу фоновых процессов и программ обработки прерываний. Тем не менее, многозадачная версия ISaGRAF предпочтительнее.

#### ═ **Конфигурирование коммуникаций: -t ключ**

Однозадачная версия ISaGRAF использует последовательный порт для связи с отладчиком. Название порта определяется при помощи ключа -t.

**Нет значения по умолчанию:** Если эта опция не использована, то связь с целевой задачей невозможна. В таком случае может быть выдано сообщение об ошибке номер 7.

В однозадачной версии связь по ETHERNET не предусмотрена.

Последовательный порт открывается в режиме передачи двоичных данных (нет управляющих данных, нет XON/XOFF). Другие параметры порта должны быть установлены до запуска ISaGRAF. При использовании отладчика убедитесь в соответствии параметров, устанавливаемых для него и для целевой задачи.

Пример:

```
xtmode /t0 baud=19200
```

Установить скорость передачи 19200 бит/с. на устройстве /t0.

#### ═ **Номер подчиненного: -s ключ**

Эта опция определяет номер целевой задачи. Он может принимать значения от 1 до 255 за исключением 13 (\$0D). Этот номер используется в протоколе связи. Он нужен для того, чтобы различать целевые задачи друг от друга, когда запущено несколько задач. Когда используется отладчик, убедитесь, что его подчиненный параметр соответствует номеру целевой задачи.

**По умолчанию** номер подчиненного равен 1.

#### ═ **Примеры:**

**isa -t=/t0** запуск однозадачного режима ISaGRAF как подчиненный 1 (по умолчанию) с портом связи /t0

**isa -s=3 -t=/t1** запуск однозадачного режима ISaGRAF как подчиненный 3 с портом связи /t1

**isa -t=/t0 & isa -s=3 -t=/t1** запуск двух задач ISaGRAF в однозадачном режиме: первая - подчиненный 1, портом /t0, вторая - подчиненный 3, порт /t1

## C.4.2 ISaGRAF в многозадачном режиме: isaker, isatst, isanet

Для того чтобы улучшить время реакции целевой задачи ISaGRAF, целевая задача разделена на два процесса: программа связи (isatst или isanet) и прикладная целевая задача (isaker).

Такая архитектура более гибкая. Она позволяет запускать коммуникационных несколько задач с одной и той же целевой задачей или до 4 целевых задач с одной и той же коммуникационной задачей.

Это также облегчает интеграцию различных процессов (например, визуализацию и отладку) в рамках одного приложения или связь через один и тот же порт с 4 приложениями.

Целевая и коммуникационная задача не зависят друг от друга. Единственное требование состоит в том, чтобы задача isaker была запущена первой, так чтобы она смогла инициализировать систему и задача связи смогла связаться с ней. Задача ISaGRAF не нарушает работу фоновых процессов и программ обработки прерываний.

### C.4.2.1 Запуск целевой задачи: isaker

#### ≡ **Номер подчиненного: -s ключ**

Эта опция определяет номер целевой задачи. Он может принимать значения от 1 до 255, исключая 13 (\$0D). Этот номер используется в протоколе связи. Он нужен для того, чтобы различать целевые задачи друг от друга, когда запущено несколько задач.

**По умолчанию** номер подчиненного равен 1 (как и в системе разработки).

### C.4.2.2 Запуск задачи связи по последовательному каналу: isatst

#### ≡ **Конфигурирование коммуникаций: -t ключ**

Коммуникационная задача isatst использует последовательный порт для связи с отладчиком. Название порта определяется при помощи ключа -t.

**Нет значения по умолчанию:** Если эта опция не использована, то связь с целевой задачей невозможна. В таком случае может быть выдано сообщение об ошибке номер 7.

При реализации, использующей isatst связь через Ethernet невозможна.

Последовательный порт открывается в режиме передачи двоичных данных (нет управляющих данных, нет XON/XOFF). Другие параметры порта должны быть установлены до запуска ISaGRAF. При использовании отладчика убедитесь в соответствии параметров, устанавливаемых для него и для целевой задачи.

Пример:

xmode /t0 baud=19200

Установить скорость передачи 19200 бит/с. на устройстве /t0.

#### ═ **Номер подчиненного: -s ключ**

Эта опция определяет номер целевой задачи. Он может принимать значения от 1 до 255, исключая 13 (\$0D). Этот ключ может быть повторен в командной строке до 4 раз (по числу подчиненных, работающих через данную задачу). Этот номер используется в протоколе связи. Он нужен для того, чтобы различать целевые задачи друг от друга, когда запущено несколько задач. При использовании отладчика убедитесь в соответствии параметров, устанавливаемых для него и для целевой задачи.

**По умолчанию** номер подчиненного равен 1.

#### ═ **Логический номер задачи связи: -c**

Эта опция определяет логический номер задачи связи. Он используется для того, чтобы работать одновременно с несколькими задачами связи. Он принимает значения от 1 до 255 и должен быть различным у различных задач.

**По умолчанию:** используется значение последнего из ключей -s. Тем самым обеспечивается совместимость с предыдущей версией ISaGRAF 3.0.

### C.4.2.3 Запуск задачи связи по ETHERNET: isanet

#### ═ **Конфигурирование коммуникаций: -t ключ**

Задача isanet использует ETHERNET для связи с отладчиком. Номер порта определяется при помощи опции -t.

**Нет значения по умолчанию:** Если эта опция не использована, то связь с целевой задачей невозможна. В таком случае может быть выдано сообщение об ошибке номер 7.

При использовании отладчика убедитесь в соответствии параметров.

Для ISaGRAF целевая задача OS-9 является сервером, а отладчик - клиентом, который подсоединяется к указанному порту. Прежде чем начать первую сессию отладки по Ethernet необходимо убедиться, что сетевые средства OS-9 сконфигурированы правильно.

#### ═ **Номер подчиненного: -s ключ**

Эта опция определяет номер целевой задачи. Он может принимать значения от 1 до 255, исключая 13 (\$0D). Этот ключ может быть повторен в командной строке до 4 раз (по числу подчиненных, работающих через данную задачу). Этот номер используется в протоколе связи. Он нужен для того, чтобы различать целевые задачи друг от друга, когда запущено несколько задач. Когда используется отладчик, убедитесь, что его подчиненный параметр соответствует номеру целевой задачи.

**По умолчанию** номер подчиненного равен 1.

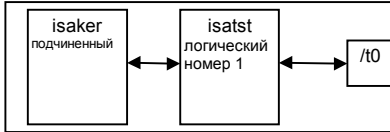
#### ═ **Логический номер задачи связи: -c**

Эта опция определяет логический номер задачи связи. Он используется для того, чтобы работать одновременно с несколькими задачами связи. Он принимает значения от 1 до 255 и должен быть различным у различных задач.

**По умолчанию:** используется значение последнего из ключей -s. Тем самым обеспечивается совместимость с предыдущей версией ISaGRAF 3.0.

#### C.4.2.4 Примеры:

**isaker &  
isatst -t=/t0**

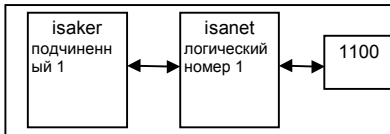


Эти команды запускают:

Ядро ISaGRAF с номером подчиненного 1 по умолчанию

Коммуникационную задачу через последовательный порт на /t0 с номером подчиненного 1 (по умолчанию) и логическим номером 1 (последний указанный подчиненный=подчиненный по умолчанию=1)

**isaker &  
isanet -t=1100**

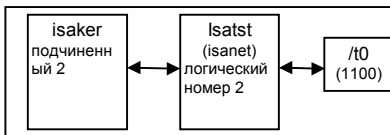


Starts:

Ядро ISaGRAF с номером подчиненного 1 по умолчанию

Коммуникационную задачу по Ethernet через порт 1100 с номером слева 1 (по умолчанию) и логическим номером 1 (последний указанный подчиненный=подчиненный по умолчанию=1)

**isaker -s=2 &  
isatst -t=/t0 -s=2** (Соответственно **isanet -t=1100 -s=2**)

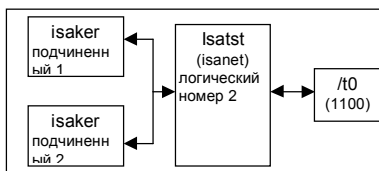


Эти команды запускают:

Ядро ISaGRAF с номером подчиненного 2

Коммуникационную задачу по последовательному порту /t0 (Ethernet порт 1100) с номером слева 2 и логическим номером 2 (последний указанный подчиненный=2)

**isaker -s=1 &  
isaker -s=2 &  
isanet -t=/t0 -s=1 -s=2** (Соответственно **isanet -t=1100 -s=1 -s=2**)



Эти команды запускают:

Ядро ISaGRAF с номером подчиненного 1

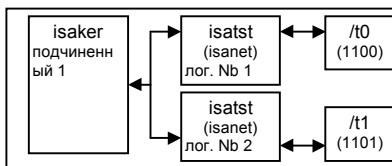
Ядро ISaGRAF с номером подчиненного 2

Коммуникационную задачу по последовательному порту /t0 (Ethernet порт 1100), взаимодействующую с подчиненными с номерами 1 и 2, и логическим номером 2 (последний указанный подчиненный=2)

**isaker -s=1 &**

**isatst -t=/t0 -s=1 -c=1 &** (Соответственно **isanel -t=1100 -s=1 -c=1 &**)

**isatst -t=/t1 -s=1 -c=2** (Соответственно **isanel -t=1100 -s=1 -c=2**)



Эти команды запускают:

Ядро ISaGRAF с номером подчиненного 1

Коммуникационную задачу по последовательному порту /t0 (Ethernet порт 1100), взаимодействующую с подчиненным номер 1, и логическим номером 1

Коммуникационную задачу по последовательному порту /t1 (Ethernet порт 1101), взаимодействующую с подчиненным номер 1, и логическим номером 2

Замечание:

Коммуникационные задачи по последовательному порту и Ethernet могут работать вместе.

### С.4.3 Специфические особенности.

#### ⇒ **Установка связи**

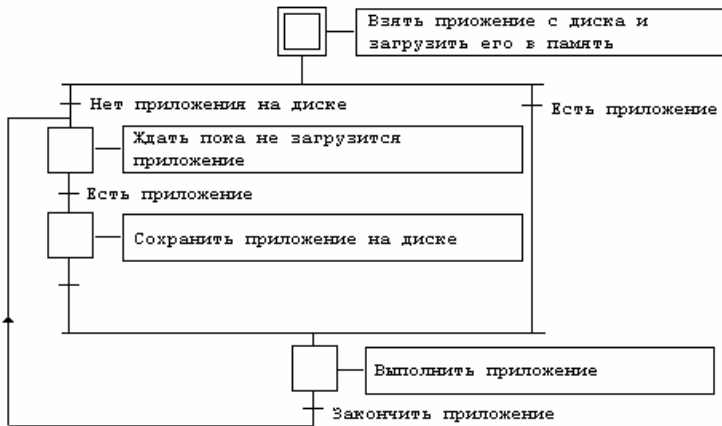
Благодаря гибкости администратора последовательных устройств OS-9 можно использовать почти любое дуплексное физическое устройство, поддерживаемое Microware.

Пример:

Последовательное устройство может быть задано указанием пути к нему, даже если это порт другого процессора. Тогда ключ -t может выглядеть следующим образом: -t/nr/MASTER/t0, где MASTER - имя процессора в ram-сети, /t0 - физический порт.

#### ⇒ **Запуск ISaGRAF.**

В начале работы ISaGRAF выполняется следующий алгоритм:



- Определения

Код приложения это двоичный код, который генерируется и загружается подсистемой разработки и затем исполняется на целевой машине. Он может быть дополнен таблицей символов.

Таблица символов - это ASCII база данных, которая генерируется и загружается подсистемой разработки. Эта таблица обеспечивает связь символьных объектов и внутренних объектов целевой задачи. Она требуется в целевой задаче только в случаях специфического управления символами.

- Объекты ISaGRAF OS-9 и многозадачные приложения

Всякое имя объекта ISaGRAF имеет префикс `ISAxn` где **x** - номер подчиненного ядра и **n** - номер, специфицирующий объект, за исключением **ISAy3**, где **y** - логический номер задачи связи в многозадачной реализации.

Различные приложения (ядра и коммуникационные задачи) могут одновременно работать на одном процессоре, коль скоро они имеют различные номера подчиненных и различные логические номера коммуникационных задач. Тем не менее, в таком режиме пользователь должен сам позаботиться об отсутствии конфликтов между приложениями, когда есть ресурсы с разделяемым доступом. Например, когда различные приложения (ядра) обращаются к физическим платам, требуется драйвер ввода/вывода или семафор.

Имена объектов в OS-9:

Дисковые файлы:

- ISAx1** резервная копия кода приложения ISaGRAF
- ISAx6** резервная копия таблицы символов приложения ISaGRAF

Модули в оперативной памяти::

ISaGRAF	<b>ISAx0</b>	данные ядра системы ISaGRAF
	<b>ISAx1</b>	код приложения ISaGRAF
	<b>ISAx2</b>	база данных реального времени ядра ISaGRAF
	<b>ISAy3</b>	буфер для обмена данными с коммуникационной задачей
	<b>ISAx4</b>	онлайн-модификация 1 кода приложения ISaGRAF
	<b>ISAx5</b>	онлайн-модификация 2 кода приложения ISaGRAF
	<b>ISAx6</b>	таблица символов приложения ISaGRAF

Пользователь не должен использовать эти имена объектов.

- **Сохранение приложения**

Когда приложение загружается отладчиком подсистемы разработки в целевую машину, код приложения сохраняется на диске в файле под именем

**ISAx1** копия кода приложения ISaGRAF (x - номер подчиненного)

Если, кроме того, прежде была загружена таблица символов, она тоже сохраняется на диске в файле под именем

**ISAx6** копия таблицы символов приложения (x - номер подчиненного, y - номер задачи связи)

Когда ISaGRAF запускается, целевая задача ищет код приложения и таблицу символов на диске и загружает их в память как модули данных с теми же именами.

Затем, если таблицы символов нет в памяти, то целевая задача исполняет код приложения без таблицы символов.

Если в памяти нет кода приложения, то целевая задача ждет, когда приложение будет загружено.

Для того чтобы запустить приложение при включении машины без использования отладчика:

Первый способ - скопируйте эти файлы на диск целевой машины с ПК, где находится система разработки. Можно использовать средства копирования системы разработки (меню Средства) для облегчения этой операции.

Второй способ - перепишите своими средствами код приложения и (если нужно) таблицу символов с ПК, где находится система разработки в долговременную память (PROM или EPROM).

При включении питания, если необходимо (для более быстрого доступа или при расстановке точек останова) можно загрузить код приложения (и, при необходимости, таблицу символов) из EPROM в RAM как **ISAx1 (ISAx6)**.

Предупреждение:

При расстановке точек останова отладчик ISaGRAF может работать некорректно, если код приложения недоступен для записи. Это не проблема, если приложение полностью протестировано.

Если на хост ПК система разработки ISaGRAF установлена в стандартном каталоге \SAWIN, то:

файл кода приложения проекта MYPROJ будет называться

\SAWIN\APL\MYPROJ\appli.x6m (соответствует ISAx1 на целевой машине)

файл таблицы символов приложения проекта MYPROJ будет называться

\SAWIN\APL\MYPROJ\appli.tst (соответствует ISAx6 на целевой машине)



## ▬ **Обработка ошибок и выдача сообщений**

Целевая задача ISaGRAF включает в себя обработчик ошибок. Список ошибок находится в конце документации.

Ошибка обрабатывается следующим образом:

- Ошибка состоит из номера и аргумента, посылаемых обработчику ошибок ISaGRAF.
- Если в системе разработки установлен флаг обнаружения ошибок, то ошибка обрабатывается. Если нет - информация теряется и обработка заканчивается.

Если обрабатывается:

- Номер (десятичное число) и аргумент (шестнадцатеричное число) выводятся в stdout.
- Номер ошибки и аргумент помещаются в кольцевой буфер FIFO, а позднее переносятся оттуда. Размер буфера определяется в опциях Make системы разработки. Когда буфер полон, сообщение о новой ошибке вытесняет самое старое сообщение.
- Системки извлекаются из буфера либо отладчиком, либо приложением, используя вызов SYSTEM (см. Руководство пользователя).

Когда отладчик фиксирует ошибку, сообщение о ней появляется в окне отображения ошибок. В зависимости от ситуации (работает приложение или нет) отладчик отображает имя объекта (переменной или программы) откуда пришла ошибка или аргумент ошибки (десятичное число) в квадратных скобках [x], который имеет различное значение для разных ошибок.

Сообщение об ошибке и ее значение по умолчанию выдаются в stdout. Если отображение в stdout нежелательно, можно использовать команду перенаправления следующего вида

```
prog_name [Опции] >>>Inil
```

## ▬ **Длительность цикла, поведение задач и их приоритеты**

- В конце каждого цикла ISaGRAF, перед тем как начать новый цикл, выполняет следующий алгоритм:

Если определено фиксированное время цикла, тогда процессор переключается на выполнение других задач на оставшийся период времени. Если оставшийся период времени отрицательный, тогда выдается сообщение о переполнении и процессор освобождается на 1 тик диспетчеризации.

Если время цикла не фиксировано или остаток времени меньше или равен 1 тик, тогда процессор освобождается на 1 тик.

Временное разрешение соответствует установленному в OS-9 размеру тика.

Указанная стратегия обычно используется для более рационального использования процессорного времени - чтобы уступить процессор другим задачам, работающим в данный момент в системе.

- Коммуникационная задача находится в спящем состоянии пока нет данных для передачи. В случае необходимости эта задача получает информацию о работающем приложении по протоколу вопрос/ответ от ядра. Коммуникационная задача запрашивает ядро. В конце цикла (чтобы успеть получить синхронный образ приложения) ядро дает ответ коммуникационной задаче.

Задачи ISaGRAF не модифицируют приоритет, который им присвоен. Пользователь волен сам регулировать приоритеты в соответствии с поведением задачи ISaGRAF и ее требованиями к системе. Например, чтобы ISaGRAF не прерывался задачами более низкого приоритета можно модифицировать такие управляющие параметры OS-9 как MIN\_AGE или MAX\_AGE.

## ▬ **Режим терминала**

Целевой последовательный протокол распознает последовательность из 3х символов возврата каретки (\$0D) и стартует задачу shell OS-9, если она доступна, на устройстве последовательной связи. Это позволяет получить строку приглашения shell на любом терминале, используя последовательную связь целевой задачи ISaGRAF.

Пример:

На хост ПК

- Закрывает отладчик ISaGRAF.
- Запустить сессию Windows Terminal (в группе accessories) с корректными коммуникационными параметрами
- Трижды нажать возврат каретки

Теперь Вы зарегистрированы в shell OS-9

- Напечатайте **logout**, чтобы выйти из режима терминала.

**ПРЕДУПРЕЖДЕНИЕ:**

Следует всегда чисто выходить из режима терминала, используя **logout**, и никак иначе. В противном случае следующее соединение с системой разработки окажется неудачным.

## C.5 Работа с целевой задачей VxWorks

Для того чтобы начать работу с целевой задачей ISaGRAF нужно выполнить несколько команд в системе VxWorks для настройки параметров среды перед запуском целевых задач ISaGRAF. Эти команды можно выполнить из файла сценария. Они описываются в следующих разделах.

### C.5.1 Администратор системных ресурсов: `isassr.o`

Этот модуль нужен всегда при любой конфигурации целевой задачи ISaGRAF и должен быть первым среди загружаемых модулей ISaGRAF. Он обеспечивает управление работающими одновременно несколькими целевыми задачами.

### C.5.2 Отличительные особенности `isa.o`, `isakerse.o` и `isakeret.o`

Чтобы запустить ISaGRAF нужно загрузить один из следующих модулей.

`isa.o`: вариант единственной целевой задачи (только последовательная связь)  
`isakerse.o`: вариант нескольких целевых задач (только последовательная связь)  
`isakeret.o`: вариант нескольких целевых задач (последовательная связь и/или по Ethernet)

Детальное описание этих модулей - в последующих разделах.

#### — **Конфигурирование последовательной связи**

Целевая задача ISaGRAF преимущественно использует последовательный порт для связи с отладчиком. При открытии целевая задача не устанавливает никаких параметров на указанный ей последовательный порт. Поэтому пользователь полностью свободен в назначении нужных ему параметров. Единственным требованием является установка бинарного режима передачи (RAW mode). Для этого предназначена функция `ISAMOD()`.

```
uchar ISAMOD
(
    char *desc,      /* Имя последовательного устройства*/
    uint32 baudrate /* Скорость передачи*/
)
```

#### Описание:

Конфигурирует указанное последовательное устройство для передачи бинарных данных с указанной скоростью.

#### Возвращаемое значение:

0 в случае успеха, BAD\_RET - в случае ошибок.

При использовании отладчика системы разработки, убедитесь, что параметры связи (см. руководство пользователя: Управление программами) соответствуют параметрам цели.

#### — **Частота системных часов**

Глобальная переменная CLKRATE (uint32) должна быть инициализирована. Это можно сделать следующим образом

```
CLKRATE = sysClkRateGet ()
```

Значение CLKRATE по умолчанию - 60 Hz.

### **C.5.3 Работа ISaGRAF в однозадачном режиме: isa.o**

ISaGRAF может работать как одна задача. В таком случае не рекомендуется перегружать коммуникации для того, чтобы гарантировать хорошую производительность. В многозадачной системе VxWorks различные целевые задачи ISaGRAF могут быть запущены на одном и том же процессоре, если они имеют различные номера подчиненных и последовательные порты.

Однозадачная реализация в основном ориентирована на слабую аппаратуру (дешевые платы и ПК, ориентированные на MS-DOS) или на разработку первого прототипа системы при портировании на новую платформу. Многозадачная реализация более предпочтительна.

Задача ISaGRAF не нарушает работу фоновых процессов и программ обработки прерываний.

#### **═ *Регистрация подчиненных***

Целевая задача ISaGRAF характеризуется номером подчиненного. Он может принимать значения от 1 до 255 за исключением 13 (\$0D). Этот номер используется в протоколе связи. Он нужен для того, чтобы различать целевые задачи друг от друга, когда запущено несколько задач. Поэтому перед запуском целевых задач следует их регистрировать. Для этого предназначена функция *isa\_register\_slave()*

```
uchar isa_register_slave  
(  
    uchar slave /* /* Номер подчиненного */  
)
```

#### Описание:

Регистрирует новый подчиненный в системе с несколькими целевыми задачами.

#### Возвращаемое значение:

0 в случае успеха, BAD\_RET - в случае ошибок.

#### **═ *Каталог для хранения резервных файлов приложения***

Глобальная переменная TSK\_FUNIT(char \*) может содержать название каталога, в котором приложение будет хранить файлы резервных копий. Целевая задача ISaGRAF использует стандартные вызовы fopen, fread, fwrite, fclose для работы с файлами.

Значение по умолчанию - пустая строка (""), указывающая, что каталог не назначен.

#### Пример:

```
TSK_FUNIT = "host name:/C:/ISaGRAF/target/apl/"
```

Назначает каталог ISaGRAF\target\apl в корне C: на хост-ПК с именем host\_name для файлов резервных копий. Важно не забыть последний слеш, иначе файлы будут открываться в каталоге ISaGRAF\target и иметь в имени префикс apl.

При необходимости эта переменная может иметь различные значения для различных целевых задач. Дальнейшие детали о файлах резервных копий содержатся в разделе Отличительные особенности / Резервирование приложения.

## ⇒ **Управление окончанием цикла**

Переменная TSK\_NBTKSCHED (uint32) имеет значение задержки в тиках, которую целевая задача ISaGRAF использует в конце цикла. Значение по умолчанию - 0 (один и тот же приоритет у всех задач). При необходимости для каждой целевой задачи это значение может быть своим.

Дальнейшие детали содержатся в разделе Отличительные особенности / Продолжительность цикла.

## ⇒ **Запуск целевой задачи ISaGRAF**

После того как установлены переменные среды, на заключительном шаге с помощью вызова isa\_main запускается целевая задача ISaGRAF:

```
uchar isa_main
(
    uchar slave,      /* Номер подчиненного*/
    char *com        /* Имя последовательного устройства*/
)
```

### Описание:

Стартует целевую задачу ISaGRAF.

### Возвращаемое значение:

Возвращает ненулевое значение в случае возникновения ошибок.

Номер подчиненного - тот же самый, который обсуждался в разделе о регистрации подчиненного. Может быть запущено несколько целевых задач при условии, что они имеют различные номера подчиненных и коммуникационных портов.

При использовании отладчика необходимо, чтобы номера подчиненных целевой задачи и системы разработки совпадали.

## ⇒ **Пример**

Данный пример показывает как запустить однозадачную реализацию целевой задачи ISaGRAF с номером подчиненного 1 и последовательным устройством связи /tyCo/1. Текущим каталогом является тот, где установлена целевая задача.

Загрузить модуль isassr.o

**ld < RELS/isassr.o**

Загрузить модуль isa.o

**ld < CMDS/isa.o**

Конфигурирование последовательной связи

**ISAMOD ("tyCo/1", 19200)**

Частота системных часов

**CLKRATE = sysClkRateGet ()**

Регистрация подчиненного

**isa\_register\_slave (1)**

Каталог для резервирования (можно опустить, ибо используется умолчание)

**TSK\_FUNIT = ""**

Управление в конце цикла (можно опустить, ибо используется умолчание)

**TSK\_NBTCKSCHED = 0**

Запуск целевой задачи ISaGRAF

**sp (isa\_main, 1, "/tyCo/1")**

#### **C.5.4 ISaGRAF в многозадачном режиме: isakerse, isakeret**

Для того, чтобы уменьшить время реакции целевой задачи ISaGRAF, целевая задача разделена на два процесса: программу связи и прикладную исполнительную задачу.

Такая архитектура более гибкая. Она позволяет запускать несколько коммуникационных задач с одной и той же целевой задачей или до 4 целевых задач с одной и той же коммуникационной задачей. Это позволяет работать через один и тот же порт с 4 целевыми задачами. Это также облегчает интеграцию различных процессов (например, визуализацию и отладку) в рамках одного приложения.

Целевая задача и задача связи не зависят друг от друга. Единственное требование состоит в том, чтобы задача ядра была запущена первой, так чтобы она смогла установить параметры системного окружения и задача связи смогла связаться с ней.

Задача ISaGraf не нарушает работу фоновых процессов и программ обработки прерываний.

Предлагается два модуля в зависимости от аппаратных возможностей:

- Для последовательной связи: isakerse.o

Этот модуль позволяет запустить ядра и последовательную связь.

- Для последовательной связи и/или через Ethernet: isakeret.o

Этот модуль позволяет запустить ядра и задачи для последовательной связи и/или для связи по Ethernet.

Способ запуска ISaGRAF с помощью этих модулей - один и тот же, с той лишь разницей, что в случае isakeret.o можно указать как имя последовательного устройства для последовательной связи, так и номер порта для связи по Ethernet, которые передаются как параметры при запуске ISaGRAF с помощью `tst_main_ex` (см. далее).

По отношению к ISaGRAF целевая задача VxWorks является сервером, а отладчик является клиентом, который подсоединяется к указываемому номеру порта.

#### **⇒ Регистрация ядра**

Ядро ISaGRAF характеризуется номером подчиненного. Он может принимать значения от 1 до 255, исключая 13 (\$0D). Этот номер используется в протоколе связи. Он нужен для того, чтобы различать целевые задачи друг от друга когда запущено несколько задач. Поэтому, прежде чем запустить ядра ISaGRAF их нужно зарегистрировать с помощью вызова `isa_register_slave()`.

```
uchar isa_register_slave
(
    uchar slave /* Номер подчиненного */
)
```

#### Описание:

Регистрирует новый подчиненный в системе с несколькими целевыми задачами.

Возвращаемое значение:

0 в случае успеха, BAD\_RET - в случае ошибок.

═ **Регистрация коммуникационных задач**

Коммуникационная задача ISaGRAF характеризуется логическим номером. Он может принимать значения от 1 до 255. Он нужен для того, чтобы различать целевые задачи друг от друга когда запущено несколько задач. Поэтому, прежде чем запустить коммуникационную задачу ISaGRAF ее нужно зарегистрировать с помощью вызова `isa_register_com()`.

```
uchar isa_register_com
(
    uchar com_id    /* Идентификатор комм. задачи */
)
```

Описание:

Регистрирует новый подчиненный в системе с несколькими целевыми задачами.

Возвращаемое значение:

0 в случае успеха, BAD\_RET - в случае ошибок.

═ **Каталог для хранения резервных файлов приложения**

Глобальная переменная `TSK_FUNIT(char *)` может содержать название каталога, в котором приложение будет хранить файлы резервных копий. Целевая задача ISaGRAF использует стандартные вызовы `open`, `fread`, `fwrite`, `fclose` для работы с файлами.

Значение по умолчанию - пустая строка (""), указывающая, что каталог не назначен.

Пример:

```
TSK_FUNIT = "host name:/C:/ISaGRAF/target/apl/"
```

Назначает каталог `ISaGRAF\target\apl` в корне C: на хост-ПК с именем `host_name` для файлов резервных копий. Важно не забыть последний слеш, иначе файлы будут открываться в каталоге `ISaGRAF\target` и иметь в имени префикс `apl`.

При необходимости эта переменная может иметь различные значения для различных целевых задач. Дальнейшие детали о файлах резервных копий содержатся в разделе Отличительные особенности / Резервирование приложения.

═ **Управление окончанием цикла**

Переменная `TSK_NBTCKSCHED (uint32)` имеет значение задержки в тиках, которую целевая задача ISaGRAF использует в конце цикла. Значение по умолчанию - 0 (один и тот же приоритет у всех задач). При необходимости для каждой целевой задачи это значение может быть своим.

Дальнейшие детали содержатся в разделе Отличительные особенности / Продолжительность цикла.

═ **Запуск задачи ядра ISaGRAF**

После того как установлены переменные среды, на заключительном шаге с помощью вызова `isa_main` запускается задача ядра ISaGRAF:

```
uchar isa_main
(
    uchar slave,    /* Номер подчиненного*/
    char *com       /* НЕ ИСПОЛЬЗУЕТСЯ, м.б. пустой строкой*/
)
```

)

Описание:

Стартует задачу ядра ISaGRAF.

Возвращаемое значение:

Возвращает ненулевое значение в случае возникновения ошибок.

Номер подчиненного - тот же самый, который обсуждался в разделе о регистрации подчиненного. Может быть запущено несколько целевых задач при условии, что они имеют различные номера подчиненных.

## ═ **Запуск коммуникационной задачи ISaGRAF**

После того как установлены переменные среды, на заключительном шаге с помощью вызова `tst_main_ex` запускается коммуникационная задача ISaGRAF:

```
uchar tst_main_ex
(
    char *com,          /* Имя коммуникационного устройства*/
    uchar *slave,      /* Указатель на 4x байтовое поле с номерами подчиненных ядер*/
    uchar com_id       /* Идентификатор коммуникационной задачи*/
)
```

Описание:

Стартует коммуникационную задачу ISaGRAF.

Возвращаемое значение:

Возвращает ненулевое значение в случае возникновения ошибок.

В 4x байтовом поле указываются номера подчиненных ядер, с которыми поддерживает связь коммуникационная задача. Если подчиненных меньше 4, соответствующие байты заполняются нулями. После того, как задача запущена это поле больше не используется.

Имя коммуникационного устройства должно совпадать с именем последовательного устройства, через которое поддерживается связь.

Можно запустить более одной коммуникационной задачи, при этом они должны иметь разные идентификаторы задач.

При использовании отладчика необходимо, чтобы номера подчиненных целевой задачи и системы разработки совпадали.

## ═ **Пример**

Данный пример показывает как запустить:

Задачу ядра ISaGRAF с номером подчиненного 1.

Коммуникационную задачу ISaGRAF с идентификатором задачи 1, связанную с ядром 1 через последовательное устройство /tyCo/1.

Коммуникационную задачу ISaGRAF с идентификатором задачи 2, связанную с ядром 1 по Ethernet через порт 1100.

Текущим каталогом является тот, где установлена целевая задача.

Загрузить модуль `isassr.o`

**ld < RELS/isassr.o**

Загрузить модуль `isakeret.o` (Его можно загрузить даже если связь по Ethernet не предполагается)

**ld < CMDS/isakeret.o**



Конфигурирование последовательной связи

**ISAMOD ("/tyCo/1", 19200)**

Частота системных часов

**CLKRATE = sysClkRateGet ()**

Регистрация подчиненного

**isa\_register\_slave (1)**

Регистрация коммуникационных задач

**isa\_register\_com (1)**

**isa\_register\_com (2)**

Каталог для резервирования (можно опустить, ибо используется умолчание)

**TSK\_FUNIT = ""**

Управление окончанием цикла (можно опустить, ибо используется умолчание)

**TSK\_NBTCKSCHEM = 0**

Запуск задачи ядра ISaGRAF

**sp (isa\_main, 1, "")**

Коммуникационная задача, связь со подчиненными

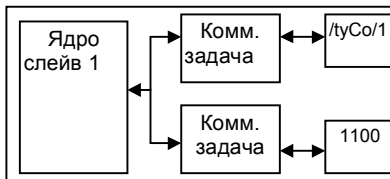
**SlavesLink = 0x01000000**

Запуск коммуникационных задач ISaGRAF

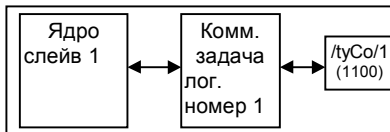
**sp (tst\_main\_ex, "/tyCo/1", &SlavesLink, 1)**

**sp (tst\_main\_ex, "1100", &SlavesLink, 2)**

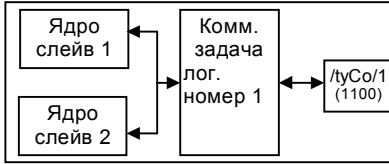
Эта последовательность соответствует следующей схеме



Возможны также следующие конфигурации.



Это наиболее типичный случай когда ядро связано с коммуникационной задачей, работающей через последовательный порт (Ethernet).

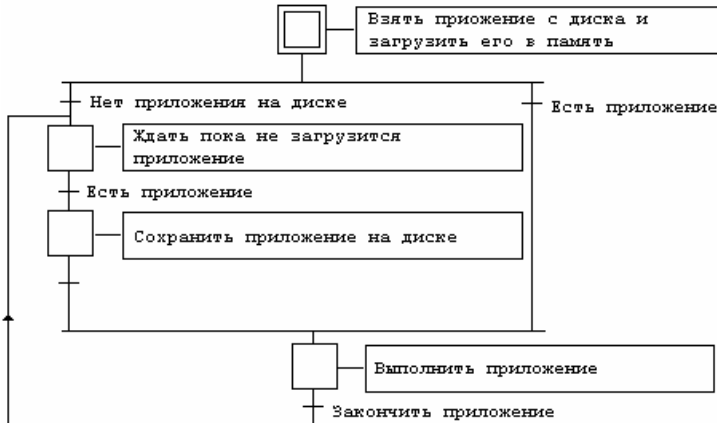


Эта конфигурация состоит из двух ядер, связанных с одной коммуникационной задачей, работающей через последовательный порт (Ethernet), SlavesLink=0x01020000.

### С.5.5 Специфические особенности.

#### — **Запуск ISaGRAF.**

В начале работы ISaGRAF выполняется следующий алгоритм:



- **Определения**

Код приложения это двоичный код, который генерируется и загружается подсистемой разработки и затем исполняется на целевой машине. Он может быть дополнен таблицей символов.

Таблица символов - это ASCII база данных, которая генерируется и загружается подсистемой разработки. Эта таблица обеспечивает связь символьных объектов и внутренних объектов целевой задачи. Она требуется в целевой задаче только в случаях специфического управления символами. По поводу таблицы символов см. в разделе Продвинутой техника программирования.

Путь к дисковым файлам прописывается в переменной TSK\_FUNIT (значение пути по умолчанию = "", т.е. нет значения).

- **Многозадачные приложения ISaGRAF**

Различные приложения (ядра и коммуникационные задачи) могут одновременно работать на одном процессоре коль скоро они имеют различные номера подчиненных и различные логические номера коммуникационных задач. Тем не менее, в таком режиме пользователь должен сам позаботиться об отсутствии конфликтов между приложениями, когда есть ресурсы с разделяемым доступом (платы V/B). Например, когда различные приложения (ядра) обращаются к физическим платам требуется драйвер ввода/вывода или семафор.

- **Сохранение приложения**

Когда новое приложение загружается отладчиком подсистемы разработки в целевую машину, код приложения сохраняется на диске (с использованием обычных вызовов для работы с файлами fopen и т.д.) в файле под именем

`path\ISAx1` копия кода приложения ISaGRAF (x - номер подчиненного)

Если, кроме того, прежде была загружена таблица символов, она тоже сохраняется на диске в файле под именем

`path\ISAx6` копия таблицы символов приложения (x - номер подчиненного)

Путь `path` указывается в переменной TSK\_FUNIT. Пустая строка ("" ) означает, что путь не указан (значение по умолчанию).

Когда ISaGRAF запускается, целевая задача ищет код приложения и таблицу символов на диске в текущем каталоге и загружает их в память.

Затем, если таблицы символов нет в памяти, то целевая задача исполняет код приложения без таблицы символов.

Если в памяти нет кода приложения, то целевая задача ждет, когда приложение будет загружено.

Для того чтобы запустить приложение при включении машины без использования отладчика:

- Первый способ - скопируйте эти файлы на диск целевой машины с ПК, где находится система разработки в предназначенный для этого каталог. Можно использовать средства копирования системы разработки (меню Инструменты) для облегчения этой операции.
- Второй способ - перепишите своими средствами код приложения и (если нужно) таблицу символов с ПК, где находится система разработки в долговременную память (PROM или EPROM).

При включении питания если необходимо (для более быстрого доступа или при расстановке точек останова) можно загрузить код приложения (и, при необходимости, таблицу символов) из EPROM в RAM.

Затем, во время старта ISaGRAF (непосредственно перед запуском задач) следует указать адреса, по которым находится код приложения и (если используется) таблица символов. Это делается с помощью инициализации глобальных переменных SSR следующим образом:

`SSR[x][1].space=указатель на адрес кода приложения`

И если необходимо

`SSR[x][6].space=указатель на адрес таблицы символов`

Таким образом вы можете написать короткую процедуру. Глобальная переменная SSR объявлена, как структура типа `str_ssr`, которая определена в файле `tasys0ssr.h`.

Предупреждение:

При расстановке точек останова отладчик ISaGRAF может работать некорректно если код приложения недоступен для записи. Это не проблема если приложение полностью протестировано.

Если система разработки ISaGRAF установлена в стандартном каталоге \ISAWIN, то:

файл кода приложения проекта MYPROJ будет называться

\ISAWIN\APL\MYPROJ\appli.x6m (соответствует isax1 на целевой машине)

файл таблицы символов приложения проекта MYPROJ будет называться

\ISAWIN\APL\MYPROJ\appli.tst (соответствует isax6 на целевой машине)

## ▬ **Обработка ошибок и выдача сообщений**

Целевая задача ISaGRAF включает в себя обработчик ошибок. Список ошибок находится в конце документации.

Ошибка обрабатывается следующим образом:

- Ошибка состоит из номера и аргумента, посылаемых обработчику ошибок ISaGRAF.
- Если в системе разработки установлен флаг обнаружения ошибок, то ошибка обрабатывается. Если нет - информация теряется и обработка заканчивается.

Если обрабатывается:

- Номер (десятичное число) и аргумент (шестнадцатеричное число) выводятся в stdout.
- Номер ошибки и аргумент помещаются в кольцевой буфер FIFO, а позднее переносятся оттуда. Размер буфера определяется в опциях Make системы разработки. Когда буфер полон, сообщение о новой ошибке вытесняет самое старое сообщение.
- Ошибки извлекаются из буфера либо отладчиком, либо приложением, используя вызов SYSTEM (см. Руководство пользователя).

Когда отладчик фиксирует ошибку, сообщение о ней появляется в окне отображения ошибок. В зависимости от ситуации (работает приложение или нет) отладчик отображает имя объекта (переменной или программы) откуда пришла ошибка или аргумент ошибки (десятичное число) в квадратных скобках [x], который имеет различное значение для разных ошибок.

Сообщение об ошибке и ее значение по умолчанию в выдаются в stdout. Таким образом, сообщения могут быть перенаправлены с помощью функции VxWorks

*ioGlobalStdSet()*

или *ioTaskStdSet()*

В последнем случае ни ядро, ни коммуникационная задача не могут генерировать ошибок.

## ▬ **Длительность цикла, поведение задач и их приоритеты**

- В конце каждого цикла ISaGRAF, перед тем как начать новый цикл, выполняет следующий алгоритм:

Если определено фиксированное время цикла, то процессор переключается на выполнение других задач на оставшийся период времени (время цикла - время цикла текущего приложения). Если оставшийся период времени отрицательный, то выдается сообщение о переполнении и процессор освобождается на TSK\_NBTCKSCHEД тиков (значение переменной устанавливается при старте ISaGRAF) для диспетчеризации.

Если время цикла не фиксировано или остаток времени меньше 1 тика или равен 0, то процессор освобождается на TSK\_NBTCKSCHEД тиков (значение переменной устанавливается при старте ISaGRAF) для диспетчеризации.

Временное разрешение соответствует установленному в VxWorks размеру тика.

Указанная стратегия обычно используется для более рационального использования процессорного времени - чтобы уступить процессор другим задачам, работающим в данный момент в системе.

- Коммуникационная задача находится в спящем состоянии пока нет данных для передачи. В случае необходимости эта задача получает информацию о работающем приложении по протоколу вопрос/ответ от ядра. Коммуникационная задача запрашивает ядро. В конце цикла (чтобы успеть получить синхронный образ приложения) ядро дает ответ коммуникационной задаче.

Задачи ISaGRAF не модифицируют приоритет, который им присвоен. Пользователь волен сам регулировать приоритеты в соответствии с поведением задачи ISaGRAF и ее требованиями к системе.

## С.6 Работа с целевой задачей ISaGRAF NT

### С.6.1 Работа ISaGRAF

В реализации под NT целевая задача представляет собой отдельную программу WISAKER.EXE, причем может работать параллельно несколько ее экземпляров. Каждый экземпляр должен иметь свой номер подчиненного.

Целевая задача не мешает работе драйверов, имеющих обработчики прерывания.

Программа WISAKER предназначена для работы с версией Windows NT 3.51 и выше.

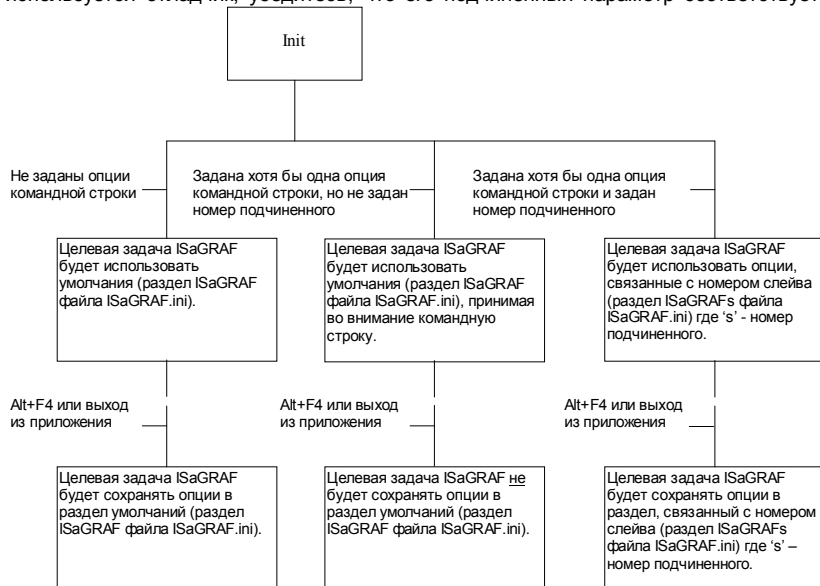
### С.6.2 Общая информация по опциям

Опции сохраняются и передаются в соответствии со следующей схемой:

Заметим, что файл ISaGRAF.INI находится в текущем рабочем каталоге.

#### — **Номер подчиненного: -s ключ**

Эта опция определяет номер целевой задачи. Он может принимать значения от 1 до 255, исключая 13 (\$0D). Этот номер используется в протоколе связи. Он нужен для того, чтобы различать целевые задачи друг от друга, когда запущено несколько задач. Когда используется отладчик, убедитесь, что его подчиненный параметр соответствует номеру



целевой задачи.

**По умолчанию** номер подчиненного равен 1 или берется из файла ISaGRAF.INI.

Пример:

WISAKER.EXE -s=2

Пользовательский интерфейс: Следующее окно открывается из основного меню целевой задачи ISaGRAF NT "Options/Slave"



Используя мышь или клавиши курсора (Up, Down) можно изменить значение. Чтобы значение вошло в силу нужно перезапустить целевую задачу ISaGRAF.

### **Конфигурирование коммуникаций: -t ключ**

Целевая задача ISaGRAF использует последовательный порт или Ethernet для связи с отладчиком. Название порта определяется при помощи ключа -t. Коммуникационный интерфейс учитывает особенности различных машин, поэтому можно использовать порты COM1, COM2, COM3 или COM4 и номера портов, начиная с 1100, для связи по Ethernet.

**Значение по умолчанию:** По умолчанию значение порта равно 1100 для связи по Ethernet и COM1 - для последовательной связи или же берется из файла ISaGRAF.INI.

ЗАМЕЧАНИЕ: По умолчанию связь устанавливается через Ethernet.

Примеры:

WISAKER -t=COM2

WISAKER -t=1101

### **Конфигурация последовательной связи:**

Некоторые опции могут быть использованы лишь при условии, что задано -t=COMx

Список опций для конфигурирования последовательной связи:

Опция	Значение	Интерпретация
baud	600	Скорость обмена
	1200	
	2400	
	4800	
	9600	
	19200	
parity	n	Нет проверки четности
	e	Четно
	o	Нечетно

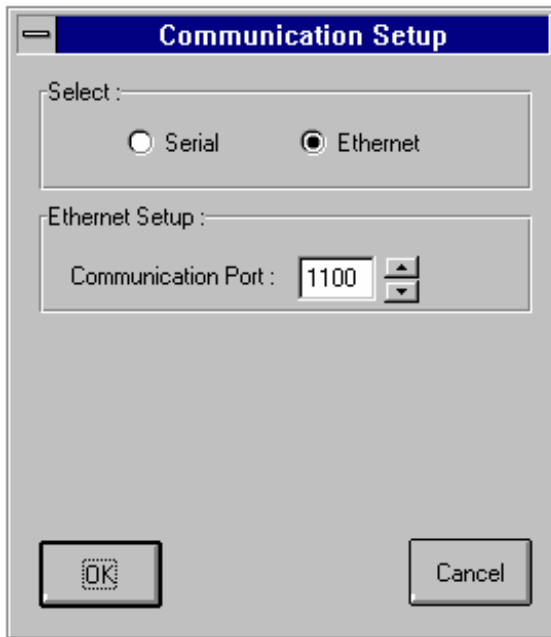
data	7 или 8	Число бит
stop	1 или 2	Число стоп бит
flow	h n	Аппаратный контроль Без контроля

Значения по умолчанию - 19200, нет проверки четности, 8 бит данных, 1 стоп бит, без аппаратного контроля.

Пример:

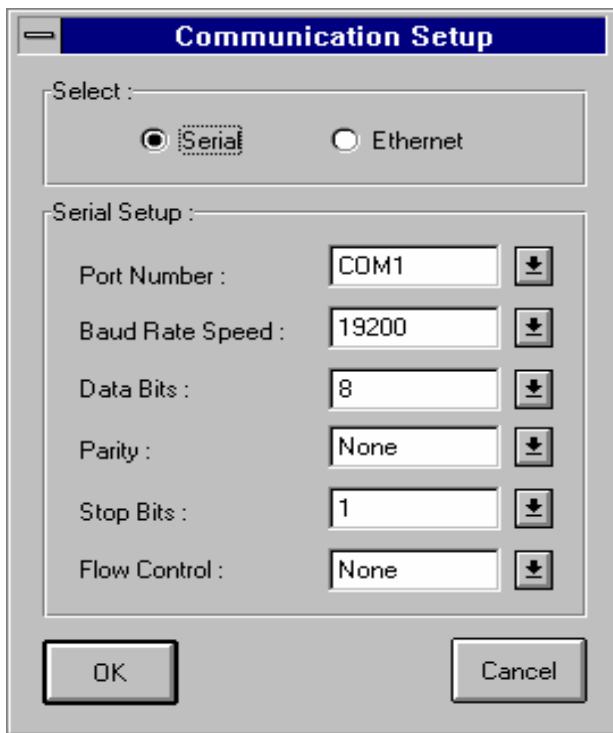
WISAKER -t=COM1 baud=1200 data=8 parity=n stop=2

Пользовательский интерфейс: Следующее окно открывается из основного меню целевой задачи ISaGRAF NT "Options/Communication"



Это дает возможность выбрать последовательную связь или связь по Ethernet. Связь по Ethernet дает возможность модифицировать номер порта. Этот номер должен быть тем же самым, который указан в системе разработки для связи между ПК и ПЛК.





При выборе последовательной связи появится окно характеристик последовательной связи. Эта конфигурация должна совпадать с той, которая дана в системе разработки для связи между ПК и ПЛК.

#### ⇒ **Графическая имитация возможных плат: -x ключ**

Если установлена эта опция, то платы, декларированные как возможные в редакторе соединений В/В, будут моделироваться.

Возможные значения 0 и 1: 0 - не моделируется, 1 - моделируется.

**Значение по умолчанию:** Значение по умолчанию - 0 или то, которое установлено в ISaGRAF.INI.

Пример:

WISAKER -x=1 будет симулировать виртуальные платы,

Интерфейс пользователя: Пункты меню помечены или не помечены, отражая состояние опции. Симулированные платы возникают на графической панели.

#### ⇒ **Приоритет целевой задачи ISaGRAF NT: -p ключ**

Для работы в среде NT полезно установить приоритетный уровень. Например, можно более критичным по времени целевым приложениям ISaGRAF присвоить более высокий приоритет, а фоновым приложениям - более низкий.

Возможные значения 0, 1, 2 или 3. 0 - высший приоритет, 3 - низший.

Примеры:

WISAKER -p=0

WISAKER -p=1

Пользовательский интерфейс: Следующее окно открывается из основного меню целевой задачи ISaGRAF NT "Options/Priority"



Высший приоритет - Real Time, низший - Idle.

0: Real Time

1: High priority

2: Normal priority

3: Idle priority

**Примеры:**

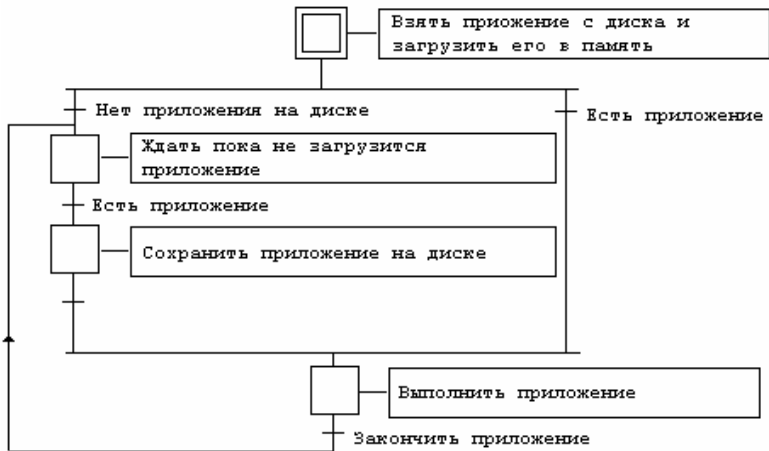
**wisaker -t=COM1** Запуск целевой задачи ISaGRAF с умолчальным номером подчиненного (1) и коммуникационным портом COM1.

**wisaker -s=3 -t=COM1** Запуск целевой задачи ISaGRAF с номером подчиненного 3 и коммуникационным портом COM1.

### С.6.3 Специфические особенности

#### **Запуск ISaGRAF**

В начале работы ISaGRAF выполняется следующий алгоритм:



- **Определения**

Код приложения это двоичный код, который генерируется и загружается подсистемой разработки и затем исполняется на целевой машине. Он может быть дополнен таблицей символов.

Таблица символов - это ASCII база данных, которая генерируется и загружается подсистемой разработки. Эта таблица обеспечивает связь символьных объектов и внутренних объектов целевой задачи. Она требуется в целевой задаче только в случаях специфического управления символами, например для DDE или для имитации В/В с использованием символьных имен. По поводу таблицы символов см. в разделе Продвинутая техника программирования.

- **Многозадачные приложения ISaGRAF**

Различные приложения могут одновременно работать на одном процессоре коль скоро они имеют различные номера подчиненных и различные логические номера коммуникационных задач. Тем не менее, в таком режиме пользователь должен сам позаботиться об отсутствии конфликтов между приложениями, когда есть ресурсы с разделяемым доступом (платы В/В). Например, когда различные приложения обращаются к физическим платам требуется драйвер ввода/вывода или семафор.

- **Сохранение приложения**

Когда новое приложение загружается отладчиком подсистемы разработки в целевую машину, код приложения сохраняется в текущем каталоге целевой задачи в файле под именем

**ISAx1**                    копия кода приложения ISaGRAF (x - номер подчиненного)

Если, кроме того, прежде была загружена таблица символов, она тоже сохраняется на диске в файле под именем

**ISAx6**                    копия таблицы символов приложения (x - номер подчиненного)

Когда ISaGRAF запускается, целевая задача ищет код приложения и таблицу символов на диске в текущем каталоге и загружает их в память.

Затем, если таблицы символов нет в памяти, то целевая задача исполняет код приложения без таблицы символов.

Если в памяти нет кода приложения, то целевая задача ждет, когда приложение будет загружено.

Для того чтобы запустить приложение при включении машины без использования отладчика эти файлы можно скопировать в текущий каталог целевой задачи с ПК, где находится система разработки.

Если система разработки ISaGRAF установлена в стандартном каталоге \ISAWIN, то:

файл кода приложения проекта MYPROJ будет называться

\ISAWIN\APL\MYPROJ\appli.x8m

файл таблицы символов приложения проекта MYPROJ будет называться

\ISAWIN\APL\MYPROJ\appli.tst

Пример:

Если из каталога, где находится WISAKER.EXE, выдать команду

copy \ISAWIN\APL\MYPROJ\appli.x8m isa11

то WISAKER.EXE будет искать и исполнять приложение тургој.

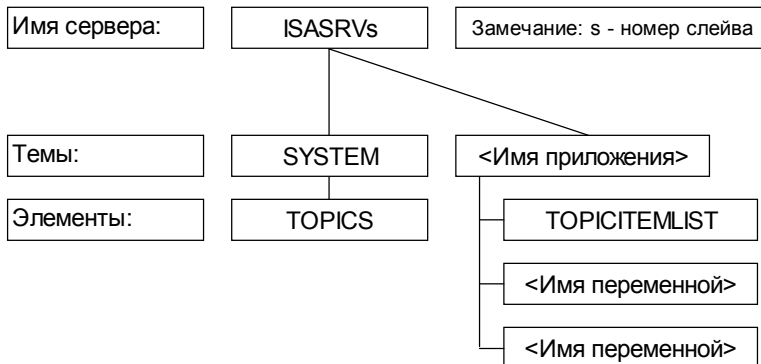
Все эти команды могут быть сгруппированы в командный файл, который будет запускаться из меню Инструменты системы разработки (см. Руководство пользователя: Управление программами)

## ⇒ Спецификации DDE

Целевая задача ISaGRAF NT является DDE (Dynamic Data Exchange) сервером. Любая программа, которая может быть клиентом, может связываться с ISaGRAF для обмена данными. Например, MSEXCEL может использоваться для анимации графики с помощью данных, приходящих от ISaGRAF по DDE.

Для использования DDE необходима таблица символов.

Субъекты DDE связаны следующим образом:



ISASRVs - имя сервера DDE, s - имя подчиненного.

SYSTEM - стандартная тема, которая дает доступ к элементу TOPICS.

TOPICS - список тем, определенных в данный момент: SYSTEM и имени приложения, которое в данный момент работает в целевой задаче ISaGRAF NT.

<Имя приложения> - имя приложения

TOPICITEMLIST - список элементов в текущей теме; он задает список переменных, которые можно получать через DDE.

<Имя переменной> - имя переменной.

#### **Режим извещения DDE: ключ -d**

Клиент DDE обычно опрашивает переменные всякий раз когда они ему нужны. Это может занять продолжительное время если переменных много. Существует другой режим, который называется режимом извещения (advise mode, advise loop), при котором сервер сам посылает только изменения. Таким образом, пересылки минимизируются. Сервер периодически просматривает переменные, для которых установлен режим извещения, чтобы узнать что нужно передавать. Этот период называется циклом извещения.

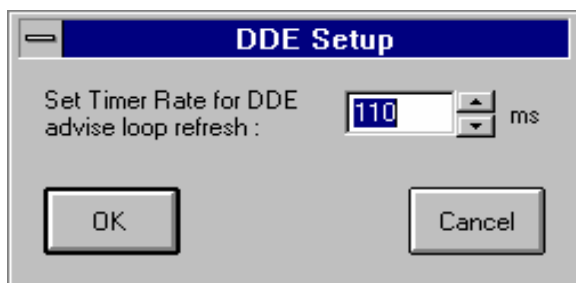
При помощи данного ключа устанавливается продолжительность цикла извещения в миллисекундах.

**Значение по умолчанию:** 1000 милисек или то, которое установлено в ISaGRAF.INI.

Пример:

WISAKER -d=100

Пользовательский интерфейс: Следующее окно открывается из основного меню целевой задачи ISaGRAF NT "Options/DDE"



#### **Обработка ошибок и выдача сообщений**

Целевая задача ISaGRAF включает в себя обработчик ошибок. Список ошибок приводится в приложении.

Обработка ошибок происходит по следующим правилам:

- Ошибка состоит из номера и аргумента, посылаемых обработчику ошибок ISaGRAF.
- Если в опциях системы разработки установлен флаг обработки ошибок, ошибки обрабатываются. Если нет - информация об ошибках просто теряется.

В случае если ошибки обрабатываются:

- Номер ошибки (десятичное число) и аргумент (шестнадцатеричное число). по умолчанию выводятся в окно WISAKER.EXE.
- Номер ошибки и аргумент помещаются в кольцевой буфер FIFO, а позднее переносятся оттуда. Размер буфера определяется в опциях Make системы разработки. Когда буфер полон, сообщение о новой ошибке вытесняет самое старое сообщение.
- Ошибки извлекаются из буфера либо отладчиком, либо приложением, используя вызов SYSTEM (см. Руководство пользователя).

Когда отладчик фиксирует ошибку, сообщение о ней появляется в окне отображения ошибок. В зависимости от ситуации (работает приложение или нет) отладчик отображает имя объекта (переменной или программы) откуда пришла ошибка или аргумент ошибки (десятичное число) в квадратных скобках [x], который имеет различное значение для разных ошибок.

Сообщение приглашения выдается когда целевая задача стартует. Оно состоит из номера подчиненного, коммуникационных конфигурационных установок и имени сервера DDE.

### ⇒ **Системные часы**

Поскольку ISaGRAF должен работать на любой системе, синхронизация цикла и обновление значений таймера происходит квантами величиной один стандартный тик (около 10 миллисекунд).

Следовательно, разрешение таймера не может быть лучше 10 мсек. По этой же причине при попытке установить продолжительность цикла менее 10 мсек. (но отличную от 0) будет выдаваться ошибка переполнения продолжительности цикла (error 62).

Обратитесь к поставщику если требуется специальная реализация, требующая лучшего разрешения.

### ⇒ **Длительность цикла и поведение целевой задач**

В конце каждого цикла ISaGRAF, перед тем как начать новый цикл, выполняет следующий алгоритм:

Если определено фиксированное время цикла, то процессор переключается на выполнение других задач на оставшийся период времени (время цикла - время цикла текущего приложения). Если оставшийся период времени отрицательный, то выдается сообщение о переполнении и процессор освобождается на 1 тик для диспетчеризации.

Если время цикла не фиксировано или остаток времени меньше или равен 1 тик или равен 0, то процессор освобождается на 1 тик для диспетчеризации.

Временное разрешение соответствует установленному в Windows NT размеру тика.

Указанная стратегия обычно используется для более рационального использования процессорного времени - чтобы уступить процессор другим задачам, работающим в данный момент в системе.

### ⇒ **Выход из системы**

При тестировании приложения в непроизводственных условиях на настольном ПК у пользователя может возникнуть необходимость остановить ISaGRAF: это делается одновременным нажатием

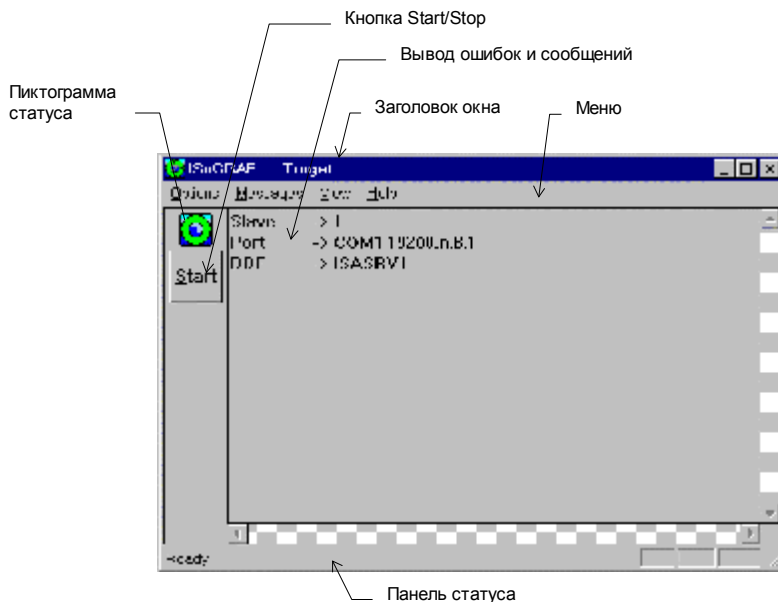
#### **alt + F4**

Один из опасных побочных эффектов быстрого выхода заключается в том, что при этом не отключается интерфейс с платами ввода/вывода. Поэтому предусмотрены аккуратные способы остановки целевой задачи ISaGRAF:

остановка приложения из отладчика (при этом интерфейс с платами в/в закрывается)  
остановки целевой задачи ISaGRAF из системного меню.

## **С.6.4 Интерфейс пользователя**

Пользовательский интерфейс целевой задачи ISaGRAF имеет вид:



Основными являются следующие элементы:

- Заголовок окна
- панель меню
- меняющаяся в зависимости от состояния пиктограмма статуса
- кнопка Start/Stop
- вывод ошибок и сообщений
- панель статуса

Заголовок окна имеет вид "ISaGRAF - name\_of\_apli - Target", где name\_of\_apli - это имя работающего приложения. Заголовок сокращается до "ISaGRAF - - Target" когда нет работающего приложения.

## ☰ **Панель меню целевой задачи ISaGRAF NT**

Панель меню имеет 4 пункта

- Options
- Messages
- View
- Help

- **Меню "Options"**

(См. также раздел Общая информация по опциям)

Меню **Options** открывает доступ к установленным для работающего приложения опциям:

**Slave** позволяет модифицировать номер подчиненного. Измененное значение вступит в силу только после рестарта целевой задачи. Эта опция - недоступна если при старте целевой задачи в командной строке была задана хотя бы одна опция.

**Communication** позволяет модифицировать конфигурацию связи. Измененное значение вступит в силу только после рестарта целевой задачи. Эта опция - недоступна если при старте целевой задачи в командной строке была задана хотя бы одна опция кроме **-s**.

**DDE** позволяет модифицировать продолжительность цикла извещения. Измененное значение вступит в силу только после рестарта целевой задачи. Эта опция - недоступна если при старте целевой задачи в командной строке была задана хотя бы одна опция кроме **-s**.

**Simulate I/O** ставит или снимает отметку в зависимости от состояния опции. Измененное значение вступит в силу только после Stop/Start приложения.

**Priority** позволяет модифицировать приоритет. Измененное значение вступает в силу немедленно.

**Default options** восстанавливает текущие значения по умолчанию для следующих параметров:

- Communication
- DDE
- координаты окна

Измененные значения вступают в силу только после рестарта целевой задачи. Эта опция - недоступна если при старте целевой задачи в командной строке была задана хотя бы одна опция кроме **-s**.

- **Меню "Messages"**

Меню "Messages" служит для управления выводом. Оно состоит из двух пунктов:

**Acknowledge** останавливает мигание красным в случае сообщений и тревог.

**Clear** полностью очищает окно вывода.





## **Пиктограмма целевой задачи ISaGRAF NT**

Пиктограмма отражает состояние целевой задачи:

- приложение работает - пиктограмма вращается
- нет приложения (или оно остановлено) - пиктограмма останавливается
- в окне вывода имеются ошибки или сообщения - центр пиктограммы мигает красным. Чтобы остановить мигание нужно выбрать Acknowledge из меню Messages или Clear из того же меню (но помните, что этот пункт полностью очищает окно). Дальнейшую информацию по ошибкам можно найти главе об обработке ошибок и выводе сообщений.



Различные состояния сведены в следующую таблицу:

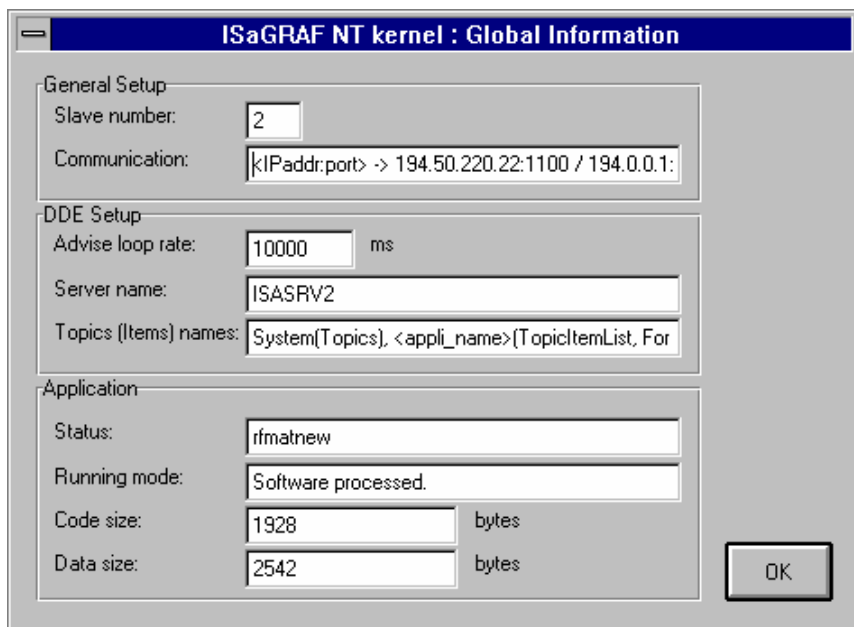
	Нет ошибок	Ошибки или сообщения (середина красная)
Запущенные приложения		
Нет приложений		

### — Кнопка Start/Stop целевой задачи ISaGRAF NT

Эта кнопка полностью аналогична такой же кнопке в отладчике. Текст на кнопке отражает текущее состояние приложения. Если приложение работает, то отображается текст Stop. Если приложение остановлено или отсутствует, то отображается текст Start (Заметьте, что если кнопка Start нажата в отсутствие приложения, ее состояние сменится на Stop и сразу же вернется на Start).

### — Целевая задача ISaGRAF NT, общая информация

С помощью команды View/Information вызывается следующее диалоговое окно, которое дает общую информацию о конфигурации целевой задачи и работающего приложения:



**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

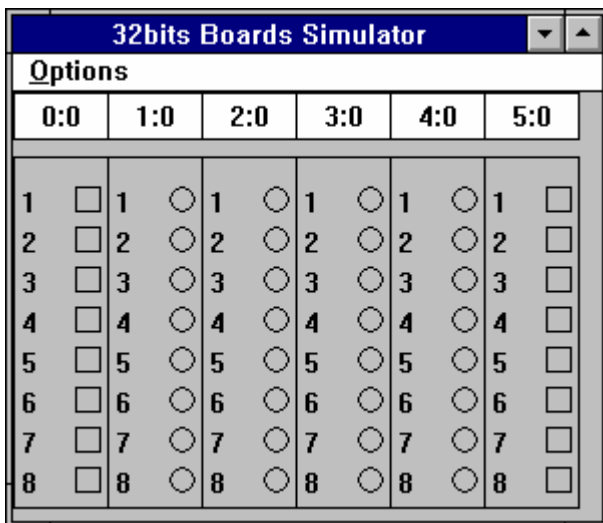
Data size:  bytes

Имеется три раздела:

- a) Общие установки
  - n Номер подчиненного
  - n Конфигурация связи (если связь по Ethernet, то в дополнение номеру порта будет отображаться список возможных адресов IP)
- b) Установки DDE
  - n продолжительность цикла оповещения
  - n имя сервера DDE
  - n имена тем и элементов DDE. Это общая информация, она не отражает реальные значения. Фактически поля между < > должны быть заменены реальными значениями.
- c) Приложение
  - n статус приложения, который представляет собой имя приложения или текст No application в отсутствие работающего приложения
  - n режим работы приложения, который указывает, было ли приложение запущено с помощью программного процессора. В этом случае графа содержит строку "Software processed". Если приложение получено с помощью компилятора C в графе появляется строка "C compiled". Если нет работающего приложения - в графе стоит строка "No application".
  - n Размер кода в байтах. Если режим "C compiled", то значение поля есть 0.
  - n Размер данных в байтах. Это суммарный размер сегмента данных исполняемого модуля и базы данных переменных.

### Имитация возможных плат целевой задачи ISaGRAF NT

Когда выбрана опция Simulate I/O при следующем старте приложения появляется следующее окно:



В зависимости от конфигурации соединений В/В Вашего приложения появится большее или меньшее число плат и большее или меньшее число переменных. Пара чисел "s:b" сверху

колонки, соответствующей каждой плате указывает на идентификатор слота (s) и идентификатор платы (b). Нумерация начинается с 0, и этот порядок изменить невозможно. Окно "32bits Boards Simulator" управляется кнопкой Start/Stop. Так, если имеется приложение с возможными платами и установлен флаг "Simulate I/O", это окно появляется. При нажатии кнопки Stop - оно исчезнет. Это окно работает вместе с вызовами В/В. Меню "Options" предлагает два пункта

**Variable names** показывает имена переменных в том (и только в том) случае, если предварительно была загружена таблица символов.

**Hexadecimal values** показывает числа в шестнадцатеричном формате вместо десятичного по умолчанию.

Имена переменных будут выглядеть следующим образом:

32bits Boards Simulator					
Options					
0:0	1:0	2:0	3:0	4:0	5:0
1 <input type="checkbox"/> Row0	1 <input type="radio"/> LED00	1 <input type="radio"/> LED10	1 <input type="radio"/> LED20	1 <input type="radio"/> LED30	1 <input type="checkbox"/> COL0
2 <input type="checkbox"/> Row1	2 <input type="radio"/> LED01	2 <input type="radio"/> LED11	2 <input type="radio"/> LED21	2 <input type="radio"/> LED31	2 <input type="checkbox"/> COL1
3 <input type="checkbox"/> Row2	3 <input type="radio"/> LED02	3 <input type="radio"/> LED12	3 <input type="radio"/> LED22	3 <input type="radio"/> LED32	3 <input type="checkbox"/> COL2
4 <input type="checkbox"/> Row3	4 <input type="radio"/> LED03	4 <input type="radio"/> LED13	4 <input type="radio"/> LED23	4 <input type="radio"/> LED33	4 <input type="checkbox"/> COL3
5 <input type="checkbox"/>	5 <input type="radio"/>	5 <input type="radio"/>	5 <input type="radio"/>	5 <input type="radio"/>	5 <input type="checkbox"/>
6 <input type="checkbox"/>	6 <input type="radio"/>	6 <input type="radio"/>	6 <input type="radio"/>	6 <input type="radio"/>	6 <input type="checkbox"/>
7 <input type="checkbox"/>	7 <input type="radio"/>	7 <input type="radio"/>	7 <input type="radio"/>	7 <input type="radio"/>	7 <input type="checkbox"/>
8 <input type="checkbox"/>	8 <input type="radio"/>	8 <input type="radio"/>	8 <input type="radio"/>	8 <input type="radio"/>	8 <input type="checkbox"/>

## С.7 Программирование на “С”

### С.7.1 Обзор

Это руководство предназначено для пользователей, уже знакомых с основными понятиями ISaGRAF и средствами системы разработки. После разработки приложения с использованием **функций преобразования, “С” функций и функциональных блоков** из стандартной библиотеки, можно разработать “пользовательские” функции преобразования, “С” функции и функциональные блоки. Это позволяет пользователю улучшать целевую задачу ISaGRAF, создавая новые библиотеки и максимально использовать возможности платформы.

С помощью системы разработки на “С”, и некоторым предыдущим опытом программирования на “С”, это руководство позволит пользователю приспособить целевую задачу ISaGRAF для наилучшего управления. Такие разработки повышают производительность целевого PLC, а также делают более удобной разработку для программиста автоматизации.

Информация, содержащаяся в этом документе, не касается какой-либо конкретной целевой системы. Некоторые особенности, тем не менее, (такие, как возможность многозадачности) не могут применяться на монозадачных системах.

#### ▬ **Стандартные возможности системы разработки ISaGRAF**

Система разработки ISaGRAF предоставляет множество функций для использования “С” библиотек при разработке систем автоматизации. Для программ автоматического управления, “С” преобразования, функции или функциональные блоки - это **“черный ящик”**, полностью определенный своим интерфейсом.

Менеджер библиотек ISaGRAF используется для того, чтобы добавлять компоненты к существующим библиотекам и определять интерфейс между “С” реализацией и этой компонентой в **ST/FBD** программе. Менеджер библиотек ISaGRAF обеспечивает, также генерацию рамочного “С” кода для преобразований, функций и функциональных блоков и включает средства для редактирования такого кода. Смотрите Руководство Пользователя ISaGRAF для более полной информации.

#### ▬ **“С” разработка**

Система разработки ISaGRAF не включает никакого компилятора “С” или кроссового средства. Пользователь должен иметь компилятор “С”, предназначенный для целевой системы ISaGRAF, чтобы интегрировать эту компоненту на “С” в ядро ISaGRAF.

Если используется кросс-компилятор, система разработки ISaGRAF открывает пользователю входы для запуска команд MS-DOS из командного файла (.bat), в окне DOS. Используемый кросс-компилятор должен идти в окне эмуляции DOS. Если нет, то Windows должен быть закрыт перед запуском компиляторов и линкеров в чистом контексте MS-DOS.

## ▬ **Технические замечания**

Менеджер Библиотек ISaGRAF позволяет пользователю создавать текстовое описание для каждой компоненты библиотеки.

Это **техническое замечание** – руководство пользователя разработанной компоненты на "С", и предназначено для программиста автоматизации, чтобы описать соответствующие преобразования, функции или функциональные блоки в приложениях ISaGRAF.

Преобразование, "С" функция или функциональный блок должны быть точно определены в техническом замечании, так чтобы программист мог реально использовать его как функцию ISaGRAF. Для "С" функции техническое замечание должно описывать:

- действие, выполняемое функцией
- полное описание параметров
- смысл возвращаемого значения
- написание параметров и возвращаемого значения
- контекст приложения

Для "С" функционального блока техническое замечание должно описывать:

- действие, выполняемое функциональным блоком
- полное описание параметров
- смысл возвращаемого значения
- написание параметров и возвращаемого значения
- контекст приложения

Для функции преобразования техническое замечание должно описывать:

- точное значение преобразования, если используется входная переменная
- точное значение преобразования, если используется выходная переменная
- пределы значения, которые могут обрабатываться

Техническое замечание может также содержать информацию о:

- полная идентификация преобразования, функции или функционального блока
- любая информация о поддержке и изменениях
- поддерживаемая целевая система
- специальные многозадачные особенности
- требуемые системные службы, память, драйверы...

### **С.7.2 "С" функции преобразований**

Система разработки ISaGRAF включает утилиту линейного преобразования для выполнения простого преобразования В/В во время работы ISaGRAF на целевой машине. Эта утилита не требует никакой "С" разработки, так как она ограничивается прямым увеличением или уменьшением непрерывных функций.

Функции преобразования позволяют пользователю применять любые сложные преобразования, с специфическими операциями описанными на языке "С". Обычно, функции преобразования определяются как для **входов, так и для выходов**. Даже если

одно из направлений не используется, его необходимо реализовать и протестировать прежде, чем интегрировать преобразование в ядро ISaGRAF, во избежание сбоя системы, обусловленного неправильным вызовом.

Функции преобразования, написанные на языке "С" компилируются и линкуются с ядром ISaGRAF. Приращенное ядро должно быть инсталлировано на целевой PLC прежде, чем функция преобразования будет использована в проекте ISaGRAF. Новая функция преобразования не может быть интегрирована в симулятор ISaGRAF. Приложения ISaGRAF нужно симулировать до ввода нестандартных функций преобразования.

Исходные коды "С" стандартных преобразований написанный CJ International инсталлируется вместе с системой разработки ISaGRAF. Они могут быть использованы как примеры для создания новых функций. Не рекомендуется **изменять** стандартные функции, так как они могут быть использованы в любом приложении ISaGRAF. Стандартные преобразования, поставляемые с системой разработки ISaGRAF поддерживаются симулятором ISaGRAF.

Предупреждение: Функции преобразования - это **синхронные** операции, вызывающиеся менеджером В/В ISaGRAF, в цикле приложения во время ввода или вывода. Время, затраченное на выполнение функции преобразования, включается во временной цикл ISaGRAF. В функции преобразования не должно быть "операций ожидания", которые неоправданно растягивают временной цикл ISaGRAF.

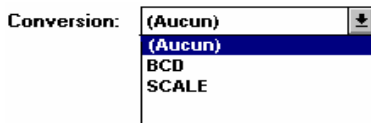
## ≡ **Добавление функции в библиотеку ISaGRAF**

Для добавления функции в библиотеку ISaGRAF нужно использовать менеджер библиотек ISaGRAF (Library Manager). Используется команда "Новый" из меню "Файлы", когда выбрана библиотека функций преобразования. Никаких параметров в системе разработки определять не надо потому, что функции преобразований используют стандартный предопределенный интерфейс.

Когда новая функция преобразования создана, должно быть написано **ее техническое замечание**. Скелет исходного текста новой функции преобразования автоматически генерируется менеджером библиотек ISaGRAF.

## ≡ **Использование преобразований в проекте ISaGRAF**

Определенные функции преобразований могут быть использованы для фильтрации входных и выходных аналоговых переменных выбранного проекта. Для того чтобы присоединить функции преобразования к переменной запускается редактор объявления переменной, выбирается входная или выходная аналоговая переменная и затем редактируются ее параметры. Поле **преобразование** диалога объявления аналога используется для установки функции преобразования присоединенной к аналоговой переменной В/В:



В списке возникают функции преобразования и таблицы. Это означает, что одно и тоже имя не может быть использовано для функции и таблицы.

Переменная не может быть присоединена к функции преобразования, которую только предполагается определить и интегрировать в ISaGRAF.

## ▬ Стандартный “С” интерфейс

Интерфейс функции преобразования всегда имеет один и тот же формат. Параметры и возвращаемое значение передаются через структуру. Структура определена в файле “TACN0DEF.h”:

```
/*
Имя: tacn0def.h
Файл определений преобразований целевой задачи
*/

#define DIR_INPUT 0          /* направление = вход преобразования */
#define DIR_OUTPUT 1       /* направление = выход преобразования */

typedef int32  T_ANA;       /* integer ANA type          */
typedef float  T_REAL;     /* real ANA type             */

typedef struct {           /* структура преобразования */
    uint16 number;        /* номер преобразования (зарезервирован) */
    uint16 direction;    /* направление преобразование */
    T_REAL *before;     /* значение до преобразования */
    T_REAL *after;      /* значение после преобразования */
} str_cnv;

#define ARG_BEFORE (*(arg->before))
#define ARG_AFTER (*(arg->after))
#define DIRECTION (arg->direction)

/* eof */
```

Структура “**str\_cnv**” полностью описывает интерфейс. Единственный параметр “С” функции преобразования - указатель на эту структуру. Поле “**number**” - это логический номер функции преобразования (положение в библиотеке ISaGRAF), оно не должно использоваться в программе.

Поле “**direction**” определяет тип переменной, к которой применяется функция преобразования (входная или выходная). Она содержит значение **DIR\_INPUT** для преобразования входов и **DIR\_OUTPUT** для преобразования выходов.

Поле “**before**” указывает на значение до преобразования. Это поле имеет различные значения для преобразований входов и преобразований выходов. Оно представляет

электрическое значение (считанное с устройства ввода) для преобразования входа, когда поле **direction** принимает значение **DIR\_INPUT**. Оно представляет физическое значение (использующееся в программе) для преобразования выхода, когда поле **direction** принимает значение **DIR\_OUTPUT**.

Поле “**after**” указывает на значение после преобразования. Это поле имеет различные значения для преобразований входов и преобразований выходов. Оно представляет физическое значение (считанное с устройства ввода) для преобразования входа, когда поле **direction** принимает значение **DIR\_INPUT**. Оно представляет электрическое значение (использующееся в программе) для преобразования выхода, когда поле **direction** принимает значение **DIR\_OUTPUT**.

Программист может использовать определения “**ARG\_BEFORE**” и “**ARG\_AFTER**” для прямого доступа к полям **before** и **after** структуры переданной в функцию преобразования. Обрабатываемые значения - **плавающие с одиночной точностью**. Результат превращается в long integer, когда преобразование применяется к целой аналоговой переменной. Это означает, что одно и тоже преобразование может быть использовано, как для целых, так и для действительных аналоговых переменных В/В.

### — **Исходный текст**

Так как функции преобразования могут быть использованы как для входных, так и для выходных аналоговых переменных, исходный “С” текст функции разделен на две основные части: преобразование входов и преобразование выходов. Поле **direction** структуры интерфейса используется для выбора преобразования, которое будет применяться. Менеджер библиотек ISaGRAF автоматически генерирует полный скелет функции, когда функция преобразования создается. Он включает основную выбирающую структуру **IF**. Ниже представлен стандартный скелет функции преобразования:

```
/*
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*/
    }
}
}
```



/\*Следующая функция показывает связь с менеджером В/В используя имя преобразования. Эта функция полностью генерируется менеджером библиотек ISaGRAF\*/

```
UFP cnvdef_sample (char *name)
{
    sys_strcpy (name, "SAMPLE");    /* дает имя преобразования */
    return (CNV_sample);          /* возвращает функцию реализации */
}
```

Лучший способ заполнить специфические части функции - это написать две отдельные локальные функции для преобразования входов и преобразования выходов. Эти функции будут вызываться основным алгоритмом, как показано в комментариях предыдущего примера, в основной структуре SFC.

Включенный файл "TASY0DEF.h" из ядра ISaGRAF требуется для системозависимых определений. Он, также содержит тип **UFP**, который представляет указатель на пустую функцию, и используется для объявления функции.

#### ▬ **Связь между проектами и "С" реализацией**

Логическая связь между реализацией функции преобразования и использованием преобразования в проекте ISaGRAF осуществляется через имя преобразования. Функция "объявления" добавляется в исходный "С" код функции преобразования. Эта функция вызывается только один раз, когда приложение стартует и сообщает менеджеру В/В имя преобразования, которое соответствует реализованной функции. Вот стандартный формат такой функции объявления:

```
UFP cnvdef_xxx (char *name)
{
    strcpy (name, "XXX");          /* имя преобразования */
    return (CNV_xxx);             /* возвращает функцию преобразования */
}
/* (xxx is the name of the conversion) */
```

Имя функции, использующееся оператором **strcpy**, должно быть написано заглавными буквами. Оно должно быть написано маленькими буквами в реализации функции преобразования и в имени функции объявления.

Использование префиксов "**CNV\_**" и "**cnvdef\_**" для реализации функции и определения функции позволяет пользователю именовать преобразования ключевыми словами языка "С" или именами существующих функций из "С" библиотек ISaGRAF.

В функцию объявления могут быть добавлены другие операторы для того, чтобы реализовать специфические операции инициализации связанные с этим преобразованием. Система ISaGRAF гарантирует, что эта функция вызывается только **однажды** при запуске приложения.

Функция объявления вызывается для любой интегрированной функции преобразования, даже если она не используется в приложении ISaGRAF. Ядро ISaGRAF выдает ошибку, если в приложении используется не реализованная функция преобразования.

Прежде чем линковать новые функции с ядром, пользователь должен написать другой файл с исходным текстом, под именем “**GRCN0LIB.C**” и вставить его с сохраненной функцией преобразования в список файлов для линкера. “**GRCN0LIB.C**” содержит только массив функций объявления. Этот массив читается во время инициализации приложения, для того чтобы создать динамические связи с функциями преобразования написанными на “С”. Вот пример такого файла:

```
/* Файл "GRCN0LIB.c" - Пример with conversions of standard library */
```

```
#include <asy0def.h>                /* required for types definition */

extern UFP cnvdef_scale (char *name); /* decl. function for SCALE conv */
extern UFP cnvdef_bcd (char *name);  /* decl. function for BCD conv */

UFP_LIST CNVDEF[ ] = {              /* array of declaration functions for */
    /* integrated conversion functions */
    cnvdef_scale,
    cnvdef_bcd,

    NULL };

/* end of file */
```

Массив **CNVDEF** должен заканчиваться указателем NULL. Если это условие не выполнено, то могут возникнуть некоторые проблемы. Если массив **CNVDEF** не определен, то во время линковки нового ядра ISaGRAF появятся неразрешенные ссылки. Написав этот файл можно построить новое ядро, включая все существующие преобразования. Можно, также построить ядро, приспособленное для одного проекта, путем введения в массив **CNVDEF** только тех преобразований, которые используются в проекте. Файл “**GRCN0LIB.C**” автоматически генерируется генератором кодов ISaGRAF, когда создается код приложения. Файл помещается в директорию проекта ISaGRAF и объединяет только те преобразования, которые используются в проекте.

### — Ограничения

Библиотека ISaGRAF может содержать до 128 функций преобразования. Функция преобразования может выполнять любой тип операции. Необходимо заметить, что функции вызываются в цикле ISaGRAF синхронно, так что выполнение функции оказывает непосредственное влияние на временной цикл.

### С.7.3 “С” функции

“С” функции используются для расширения стандартных возможностей языков **ST** и **FBD**. Они могут быть использованы для реализации специфических вычислений, системных вызовов, связи или для установки набора средств диалога между приложением ISaGRAF и другими задачами. Функции, написанные на языке “С” компилируются и линкуются с ядром

ISaGRAF. Приращенное ядро должно быть установлено на целевой PLC прежде, чем функция будет использована в проекте ISaGRAF.

Новые функции не могут быть интегрированы в симулятор ISaGRAF. Приложения ISaGRAF нужно симулировать до ввода нестандартных функций.

Предупреждение: Функции - это **синхронные** операции, вызываемые менеджером В/В ISaGRAF, в цикле приложения во время ввода или вывода. Время, затраченное на выполнение функции, включается во временной цикл ISaGRAF. В функции не должно быть "операций ожидания", которые неоправданно растягивают временной цикл ISaGRAF.

### ⇒ **Добавление функции в библиотеку ISaGRAF**

Для добавления функции в библиотеку ISaGRAF нужно использовать менеджер библиотек ISaGRAF (Library Manager). Используется команда "**Новый**" из меню "Файл", когда выбрана библиотека "С" функций. Когда новая "С" функция создана, должно быть написано **ее техническое замечание**. Скелет исходного текста новой "С" функции автоматически генерируется менеджером библиотек ISaGRAF.

Для того чтобы определить параметры вызова и возврата новой функции используется команда "**Parameters**" из меню "**Редактировать**".

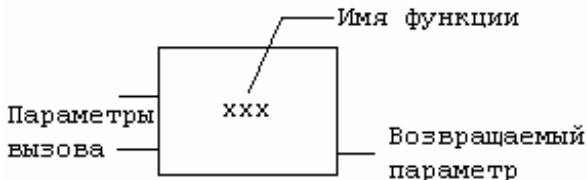
### ⇒ **Использование "С" функций в проекте ISaGRAF**

Любая интегрированная "С" функция может быть использована как стандартная функция в программах проекта ISaGRAF. "С" функции могут быть вызваны из языков **ST** и **FBD** и из специальных операторов языка **SFC**.

Вызов "С" функций из языка **ST** следует соглашениям языка по вызову функций. Параметры вызова функции записываются после имени функции, между скобок, и разделяются запятыми. Выражение представляет значение, возвращаемое функцией. Вызов "С" функции может быть введен в любой оператор присвоения или сложное выражение. Вот пример вызова "С" функции в операторе присвоения:

**result := ProcName (par1, par2, ... parN);**

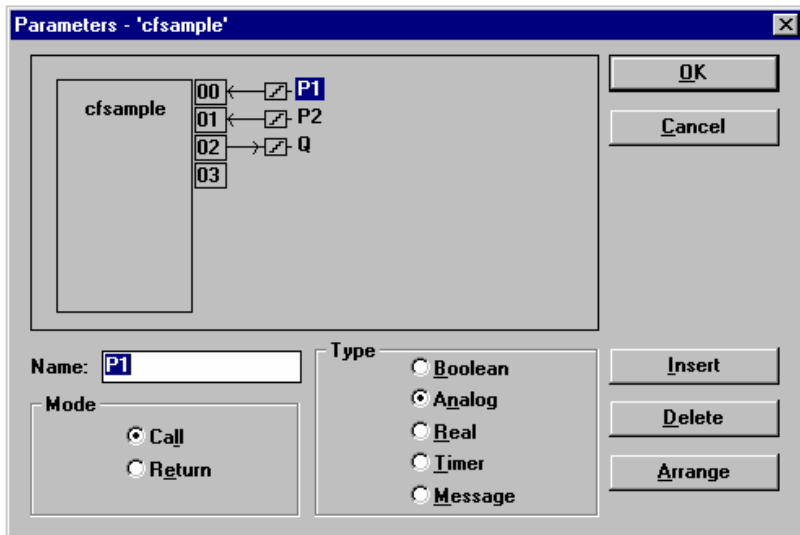
Программа **FBD** может вызывать любую "С" функцию. Функция используется как стандартный функциональный ящик. Ее параметры вызова соединяются с левой стороной функционального ящика. Возвращаемый параметр соединяется с правой стороной ящика. Вот стандартный вид такого функционального ящика:



"С" функция может быть вызвана из любого блока действия **SFC** или любого блока действия, прикрепленного к переходу.

### ⇒ **Определение интерфейса "С" функции**

Команда “**Параметры**” меню “**Редактировать**” используется для определения параметров вызова и возврата новой функции. Функция может иметь до **31** параметра вызова, и всегда имеет **один** параметр возврата. Следующий диалог используется для описания параметров “C” функции:



Список в верхней части окна показывает параметры “C” функции, в соответствии с порядком прототипа: сначала параметры вызова, в конце параметр возврата. Нижняя часть окна показывает детальное описание параметров выбранных в списке:

- имя параметра
- направление параметра (вызов/возврат)
- тип параметра

Для параметра может быть использован любой тип данных: булевский, целый аналоговый, действительный аналоговый, таймер или сообщение. Целый или аналоговый действительный должны различаться.

Ниже представлено соответствие между типами ISaGRAF и “C”:

BOOLEAN	unsigned long	32 битное беззнаковое слово: 1=true / 0=false
ANALOG	long	32 битное знаковое целое слово
REAL	float	плавающее значение с одиночной точностью
TIMER	unsigned long	32 битное беззнаковое слово (единица - это 1 миллисекунда)
MESSAGE	char *	строка символов

Когда значение сообщения приходит в “C” функцию, оно не может содержать нулевого символа. Строка, приходящая в “C” код заканчивается нулем. Не забывайте, что возвращаемый параметр должен быть последним в списке. Имена параметров должны удовлетворять следующим правилам:

- имя не может быть длиннее **16** символов

- первым символом должна быть **буква**
- последующими символами могут быть **буквы, цифры** или символ подчер
- заглавные и прописные буквы не различаются

Одно и то же имя не может быть использовано более чем для одного параметра функции. Параметр вызова не может иметь тот же тип, что и имя возвращаемого параметра. Одно и то же имя может быть использовано для параметров различных функций. Имя возвращаемого параметра по умолчанию - "Q". Это имя может быть изменено. Имя параметра используется для идентификации исходного текста "C".

Команда "Вставить" используется для ввода нового параметра перед выбранным параметром. Команда "Удалить" используется для уничтожения выбранного параметра. Команда "Упорядочить" автоматически сортирует параметры так, что возвращаемый параметр оказывается в конце списка. При нажатии кнопки "Принять" определение интерфейса функции запоминается, и диалог закрывается. Нажатие кнопки "Отказ" закрывает диалог, без изменения интерфейса функции.

### ☰ **Интерфейс "C" функции**

Интерфейс функции зависит от определения ее параметров. Параметры передаются через структуру. Эта структура определена в файле "GRUS0nnn.H", где "nnn" - это логический номер функции в библиотеке ISaGRAF. Вот пример "C" интерфейса функции "SIN" (вычисление синуса):

```

/* Файл: 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 */

```

Ниже представлено соответствие между типами ISaGRAF и "C". Типы ISaGRAF определены как типы "C" в файле определения функции:

boolean	T_BOO	long (32 бита)
Integer analog	T_ANA	long

Real analog	T_REAL	float (32 бита, плавающее значение с одиночной точностью)
timer	T_TMR	long
message	T_MSG	char* (32 битный указатель)

Каждое поле структуры "str\_srg" соответствует одному параметру функции. Возвращаемый параметр - последний в структуре. Параметры вызова возникают в структуре в том же порядке, в котором они были установлены в определении функции. Имена идентификаторов вводятся во время определения функции в менеджере библиотек ISaGRAF.

Файл определений изменяется каждый раз, когда интерфейс функций изменяется менеджером библиотек ISaGRAF. Это обеспечивает полное соответствие между реализацией функции и его использованием в программах ISaGRAF.

### — Исходный текст

Ниже представлен стандартный скелет "C" функции:

/\* Пример функции пользователя - Номер "255" - имя "SAMPLE"\*/

```
#include "tasy0def.h"           /* Общие определения ядра ISaGRAF */
#include "grus0255.h"          /* определения интерфейса для функции */
```

```
void USP_sample (str_arg *arg)
{
    /* тело функции */
}
```

/\*Следующая функция используется для инициализации функции и объявления ее реализации. Она реализует связь с ядром ISaGRAF, используя имя функции. Эта функция полностью генерируется менеджером библиотек ISaGRAF\*/

```
UFP uspdf_sample (char *name)
{
    strcpy (name, "SAMPLE"); /* дает имя функции */
    return (USP_sample);    /* возвращает реализацию функции */
}
```

/\* конец файла \*/

Включенный файл "TASY0DEF.h" из ядра ISaGRAF требуется для системозависимых определений. Он, также содержит тип **UFP**, который представляет указатель на пустую функцию, и используется для объявления функции.

### — Связь между проектами и "C" реализацией

Логическая связь между реализацией функции и ее использованием в программах проекта ISaGRAF осуществляется через имя функции. Функция объявления добавляется в исходный "C" код функции. Эта функция вызывается только один раз, когда приложение стартует и сообщает менеджеру В/В имя, которое соответствует реализованной функции. Вот стандартный формат такой функции объявления:

```
UFP uspdev_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) */
```

Имя функции, используемое оператором **strcpy**, должно быть написано заглавными буквами. Оно должно быть написано маленькими буквами в реализации функции и в имени функции объявления. Использование префиксов "USP\_" и "uspdev\_" для реализации функции и определения функции позволяет пользователю именовать функции ключевыми словами языка "C" или именами существующих функций из "C" библиотек ISaGRAF.

В функцию объявления могут быть добавлены другие операторы для того, чтобы реализовать специфические операции инициализации связанные с этой функцией. Система ISaGRAF гарантирует, что эта функция вызывается только **однажды** при запуске приложения. Функция объявления вызывается для любой интегрированной функции, даже если она не используется в приложении ISaGRAF. Ядро ISaGRAF выдает ошибку, если в приложении используется не реализованная функция.

Прежде чем линковать новые функции с ядром, пользователь должен написать другой файл с исходным текстом, под именем "GRUS0LIB.C" и вставить его с сохраненной функцией в список файлов для линкера. "GRUS0LIB.C" содержит только массив функций объявления. Этот массив читается во время инициализации приложения, для того чтобы создать динамические связи с функциями написанными на "C". Вот пример такого файла:

```
/* Файл "GRUS0LIB.c" - Пример using trigonometric functions */

#include <tasy0def.h>          /* required for types definition */

extern UFP uspdev_fc1 (char *name);    /* declaration functions */
extern UFP uspdev_fc2 (char *name);
extern UFP uspdev_fc3 (char *name);
extern UFP uspdev_fc4 (char *name);

UFP_LIST USPDEF[ ] = {      /* array of declaration functions */
    /* for integrated functions */
    uspdev_fc1,
    uspdev_fc2,
    uspdev_fc3,
    uspdev_fc4,
```

```
NULL };
```

```
/* end of file */
```

Массив **USPDEF** должен заканчиваться указателем NULL. Если это условие не выполнено, то могут возникнуть некоторые проблемы. Если массив **USPDEF** не определен, то во время линковки нового ядра ISaGRAF появятся неразрешенные ссылки. Написав этот файл, можно построить новое ядро, включая все существующие функции. Можно, также построить ядро, приспособленное для одного проекта, путем введения в массив **USPDEF** только тех преобразований, которые используются в проекте. Файл “**GRUSOLIB.C**” автоматически генерируется генератором кодов ISaGRAF, когда создается код приложения. Файл помещается в директорию проекта ISaGRAF и объединяет только те функции, которые используются в проекте.

### ⇒ **Ограничения**

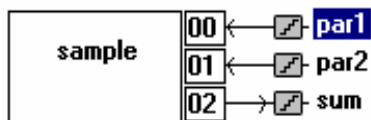
Библиотека ISaGRAF может содержать до 255 “C” функций. Функция может выполнять любой тип операции. Необходимо заметить, что функции вызываются в цикле ISaGRAF **синхронно**, так что выполнение функции оказывает непосредственное влияние на временной цикл.

### ⇒ **Полный пример**

Техническое замечание:

имя:	Sample
описание:	счетчик вверх
дата:	18 мая 1995
автор:	CJ International
вызов:	CU: считающийся вход R: команда сброса PV: максимальная планируемая величина
возврат:	Q: определение максимума CV: результат счета
прототип:	sample (count, reset_command, maximum_value); max_detect := SAMPLE.Q; count_result := SAMPLE.CV;

Интерфейс функции:



Заголовок функции:



```
/* Файл: GRUS0255.h - user C function definitions - Name: sample */
```

```
/* definition of standard ISaGRAF data types */
```

```
typedef long T_BOO;  
typedef long T_ANA;  
typedef float T_REAL;  
typedef long T_TMR;  
typedef char *T_MSG;
```

```
/* definition of the calling and return parameter structure */
```

```
typedef struct {  
    T_ANA _par1;           /* calling parameter #1 */  
    T_ANA _par2;           /* calling parameter #2 */  
    T_ANA _sum;           /* return parameter */  
} str_arg;
```

```
/* identifiers used to access call and return parameters */
```

```
#define PAR1                (arg->_par1)  
#define PAR2                (arg->_par2)  
#define SUM                  (arg->_sum)
```

```
/* end of file */
```

Ниже приведен исходный "C" код функции. Программист вручную вводит только те строки, которые набраны жирным шрифтом.

```
/* Файл: GRUS0255.c - user C function - Name: SAMPLE */
```

```
#include "tasy0def.h"           /* required for types definition */  
#include "grus0255.h"          /* C function source header */
```

```
/* C main service: calculates the addition */
```

```
void USP_sample (str_arg *arg)  
{  
    SUM = PAR1 + PAR2;  
}
```

```
/* declaration service required for dynamic link with ISaGRAF kernel */
```

```
UFP uspdf_sample (char *name)
{
    strcpy (name, "SAMPLE");
    return (USP_sample);
}
/* end of file */
```

### С.7.4 “С” функциональные блоки

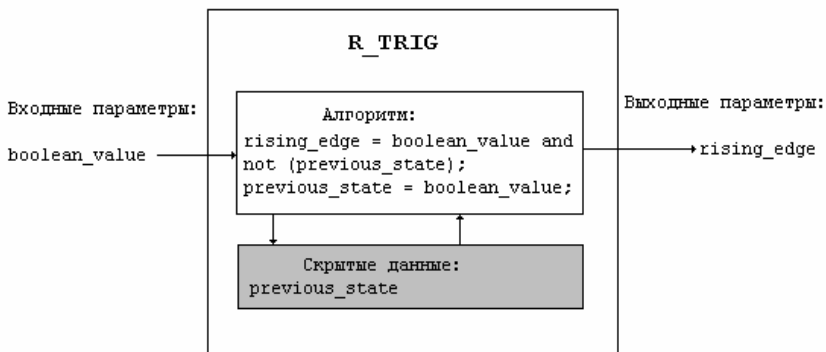
“С” функциональные блоки объединяют операции и статические данные. Они дополняют набор “С” функций, допуская обработку статических объектов. “С” функции, обычно, используются для расширения стандартных возможностей языков **ST** и **FBD**. В отличие от функций, они могут работать со статическими данными. Это означает, что алгоритм функционального блока может меняться с течением времени.

Функциональные блоки, написанные на языке “С” компилируются и линкуются с ядром ISaGRAF. Приращенное ядро должно быть установлено на целевой PLC прежде, чем функция будет использована в проекте ISaGRAF. Новые функциональные блоки не могут быть интегрированы в симулятор ISaGRAF. Приложения ISaGRAF нужно симулировать **до** ввода нестандартных функций.

Предупреждение: Функциональные блоки выполняют **синхронные** операции, вызывающиеся ядром ISaGRAF, в цикле приложения. Время, затраченное на выполнение функции, включается во **временной цикл** ISaGRAF. В функциональном блоке не должно быть “операций ожидания”, которые неоправданно растягивают временной цикл ISaGRAF.

#### — **Объявление экземпляров функционального блока**

Функциональный блок - это объект, который объединяет операции и статические данные. Ниже представлен пример функционального блока **R\_TRIG**, который определяет передний фронт булевского выражения. Вот его описание:



Скрытая статическая переменная “**previous\_state**” нужна для определения фронта. Эта переменная должна быть различной в каждом использовании функционального блока “R\_TRIG” в приложении. Экземпляр функционального блока, используемого в языке ST, должен быть объявлен в словаре. Так как функциональный блок имеет внутренние скрытые данные, каждая копия (экземпляр) функционального блока должна быть идентифицирована уникальным именем. Наименование типа блока осуществляется при помощи менеджера библиотек. Наименование экземпляров осуществляется при помощи редактора словаря.

Функциональные блоки, использующиеся в языке FBD не нужно объявлять, потому что редактор FBD автоматически объявляет экземпляры используемых блоков. Экземпляры функциональных блоков автоматически объявленные редактором FBD всегда **локальные** для редактируемой программы.

### **▬ Добавление функционального блока в библиотеку ISaGRAF**

Для добавления функционального блока в библиотеку ISaGRAF нужно использовать менеджер библиотек ISaGRAF. Используется команда “**Новый**” из меню “Файл”, когда выбрана библиотека функциональных блоков. Когда новый функциональный блок создан, должно быть написано **его техническое замечание**. Скелет исходного текста нового функционального блока автоматически генерируется менеджером библиотек ISaGRAF. Для того чтобы определить параметры вызова и возврата нового функционального блока используется команда “**Параметры**” из меню “**Редактировать**”.

### **▬ Добавление функционального блока в библиотеку ISaGRAF**

Для добавления функционального блока в библиотеку ISaGRAF нужно использовать менеджер библиотек ISaGRAF (Library Manager). Используется команда “**Новый**” из меню “Файл”, когда выбрана библиотека функциональных блоков. Когда новый функциональный блок создан, должно быть написано **его техническое замечание**. Скелет исходного текста нового функционального блока автоматически генерируется менеджером библиотек ISaGRAF. Для того чтобы определить параметры вызова и возврата нового функционального блока используется команда “**Parameters**” из меню “**Редактировать**”.

Вызов функционального блока из языка **ST** следует соглашениям языка по вызову функциональных блоков. Параметры вызова функционального блока записываются после имени функции, между скобок, и разделяются запятыми. Параметры возврата можно брать по одному. Каждый возвращаемый параметр представляется именем, объединяющим имя экземпляра блока и имя параметра. Компоненты имени разделяются точкой. Например:

**FBINSTANCE.paname**

используется, чтобы представить возвращаемый параметр “**paname**”, экземпляра функционального блока “**FBINSTANCE**”.

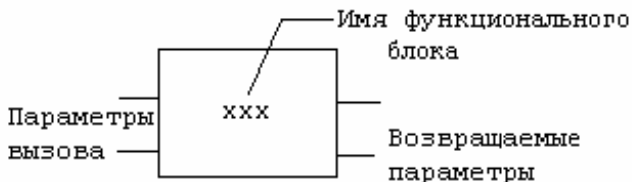
Экземпляры функционального блока, используемого в языке ST, должны быть объявлены в словаре. Каждая копия (экземпляр) функционального блока должна быть идентифицирована уникальным именем. Вот пример объявлений в библиотеке ISaGRAF:

instance:	TRIG1 TRIG2	type:	R_TRIG R_TRIG
-----------	----------------	-------	------------------

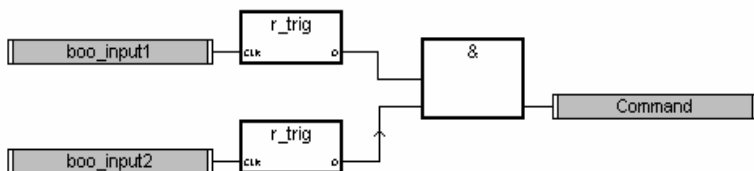
И пример использования, объявленных экземпляров в ST программе:

```
TRIG1 (boo_input1);
TRIG2 (boo_input2);
Command := (TRIG1.Q & TRIG2.Q);
```

Программа **FBD** может вызывать любой функциональный блок. Функциональный блок используется как стандартный функциональный ящик. Его параметры вызова соединяются с левой стороной функционального ящика. Возвращаемые параметры соединяются с правой стороной ящика. Вот стандартный вид такого функционального ящика:



Функциональные блоки, используемые в языке FBD не нужно объявлять, потому что редактор FBD автоматически объявляет экземпляры используемых блоков. Экземпляры функциональных блоков, автоматически объявленные, редактором FBD всегда **локальные** для редактируемой программы. Ниже пример программы на языке FBD:



### Определение интерфейса “С” функционального блока

Команда “**Параметры**” меню “**Edit**” используется для определения параметров вызова и возврата нового функционального блока. Функциональный блок может иметь до **32** параметров вызова или возврата. В отличие от “С” функции, функциональный блок может иметь несколько возвращаемых параметров. Следующий диалог используется для описания параметров “С” функционального блока:

Список в верхней части окна показывает параметры “С” функционального блока, в соответствии с порядком прототипа: сначала параметры вызова, в конце параметры возврата. Нижняя часть окна показывает детальное описание параметров выбранных, в настоящее время, в списке:

- имя параметра
- направление параметра (вызов/возврат)
- тип параметра

Для параметра может быть использован любой тип данных: булевский, целый аналоговый, действительный аналоговый, таймер или сообщение. Целый или аналоговый действительный должны различаться.

Ниже представлено соответствие между типами ISaGRAF и "C":

BOOLEAN	unsigned long	32 битовое беззнаковое слово: 1=true / 0=false
ANALOG	long	32 битное знаковое целое слово
REAL	float	плавающее значение с одиночной точностью
TIMER	unsigned long	32 битное беззнаковое слово (единица - это 1 миллисекунда)
MESSAGE	char *	строка символов

Когда значение сообщения приходит в "C" функцию, оно не может содержать нулевого символа. Строка приходящая в "C" код заканчивается нулем. Не забывайте, что возвращаемые параметры должны быть последними в списке. Имена параметров должны удовлетворять следующим правилам:

- имя не может быть длиннее **16** символов
- первым символом должна быть **буква**
- последующими символами могут быть **буквы, цифры** или символ подчеркивания
- заглавные и прописные буквы не различаются

Одно и то же имя не может быть использовано более чем для одного параметра функционального блока. Параметр вызова не может иметь тот же тип, что и имя возвращаемого параметра. Одно и то же имя может быть использовано для параметров различных функциональных блоков. Имя возвращаемого параметра по умолчанию - "**Q**". Это имя может быть изменено. Имя параметра используется для идентификации исходного текста "C".

Команда "**Вставить**" используется для ввода нового параметра перед выбранным параметром. Команда "**Удалить**" используется для уничтожения выбранного параметра. Команда "**Упорядочить**" автоматически сортирует параметры так, что возвращаемые параметры оказываются в конце списка. При нажатии кнопки "**Принять**" определение интерфейса функционального блока запоминается, и диалог закрывается. Нажатие кнопки "**Отказ**" закрывает диалог, без изменения интерфейса функционального блока.

### ☞ **Интерфейс "C" функционального блока**

Интерфейс функционального блока зависит от определения его параметров. Параметры передаются через структуру. Эта структура определена в файле "**GRFB0nnn.H**", где "**nnn**" - это логический номер функции в библиотеке ISaGRAF. Вот пример "C" интерфейса функционального блока "**LIM\_ALARM**" (тревога предела):

```
/* function block interface - name: sample */
```

```
/* standard ISaGRAF data types */
```

```
typedef long T_BOO;
typedef long T_ANA;
typedef float T_REAL;
typedef long T_TMR;
typedef char *T_MSG;
```

```

/* structure of calling parameters */

typedef struct {
    /* CALL */          T_BOO _par1;
    /* CALL */          T_BOO _par2;
} str_arg;

/* access to fields of str_arg structure */

#define                PAR1  (arg->_par1)
#define                PAR2  (arg->_par2)

/* return parameter logical numbers */

#define                FBLPNO_Q1  0
#define                FBLPNO_Q2  1

/* конец файла*/

```

Ниже представлено соответствие между типами ISaGRAF и "C". Типы ISaGRAF определены как типы "C" в файле определения функционального блока:

boolean	T_BOO	long (32 бита)
analog	T_ANA	long
real	T_REAL	float (32 бита, плавающее значение с одиночной точностью)
timer	T_TMR	long
message	T_MSG	char* (32 битный указатель)

Каждое поле структуры "str\_srg" соответствует одному параметру функционального блока. Параметры вызова в структуре в том же порядке, в котором они были установлены в определении функционального блока. Идентификатор, написанный заглавными буквами, определяется для того, чтобы иметь прямой доступ к параметрам структуры передающейся в "C" реализацию службы активизации функционального блока. Имена идентификаторов вводятся во время определения функции в менеджере библиотек ISaGRAF.

Порядок нумерации возвращаемых параметров тот же который был установлен в определении функционального блока. Логический номер первого возвращаемого параметра всегда равен 0.

Определенные идентификаторы нужно использовать вместо численных значений для того, чтобы представить возвращаемые параметры в исходном тексте "C" программы. Это

гарантирует, что исходный файл может быть легко перекомпилирован после изменений интерфейсных определений.

Файл определений изменяется каждый раз, когда интерфейс функций изменяется менеджером библиотек ISaGRAF. Это обеспечивает полное соответствие между реализацией функции и его использованием в программах ISaGRAF.

### ▬ **Исходный текст**

Реализация функционального блока на “С” делится на три входные точки:

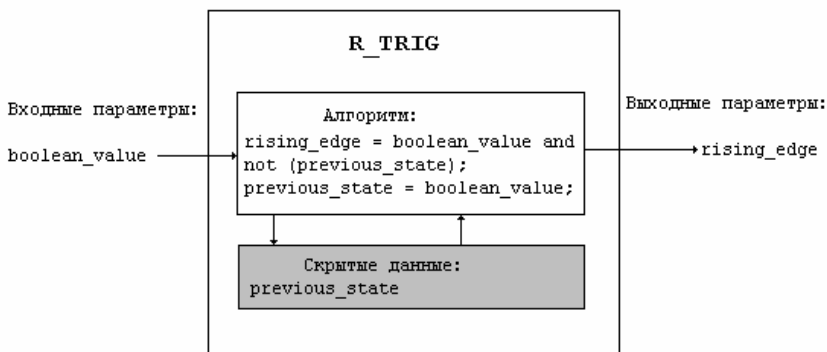
- службы инициализации
- службы активизации - обработка параметров вызова
- службы чтения возвращаемых параметров

Один и тот же код используется для каждого экземпляра одного функционального блока и не дублируется. С каждым экземпляром связана структура статических данных. К этим данным нет прямого доступа из ISaGRAF. Они содержат “скрытые переменные” функционального блока.

“Служба активизации” вызываются однажды для каждого экземпляра каждого использующегося блока, на каждом цикле целевой задачи. Она обрабатывает параметры вызова и изменяет соответствующие данные. Она представляет “основной алгоритм” функционального блока.

“Служба чтения” вызывается ядром ISaGRAF для чтения текущего значения одного возвращаемого параметра для одного экземпляра. В такой службе не нужно выполнять никаких специальных вычислений. Она только осуществляет передачу между скрытыми данными и приложением ISaGRAF.

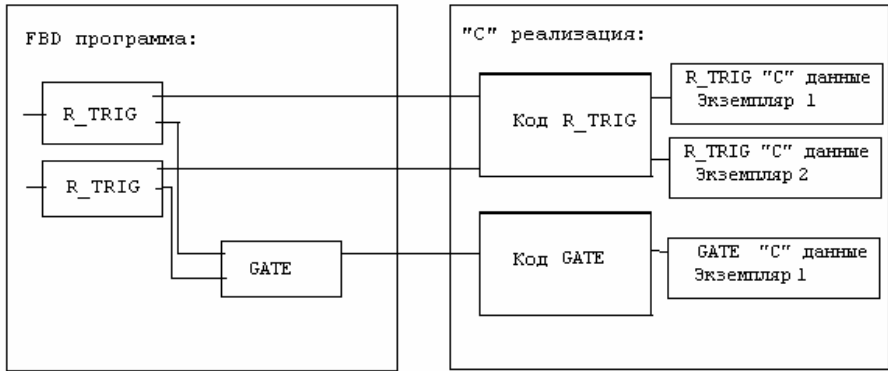
Функциональная схема:



- **Статические данные функционального блока**

Функциональный блок связывает операции и статические данные. С каждым экземпляром одного и того же функционального блока связана структура данных. Каждый раз, когда функциональный блок используется в программах ST или FBD, он соответствует одному

экземпляру и одной структуре данных. Следующий пример показывает соответствие между структурами данных "С" и экземплярами функционального блока, использующимися в FBD программе:



Память, требующаяся для структуры данных каждого экземпляра, размещается системой ISaGRAF, при запуске приложения. Указатель на соответствующую структуру данных экземпляра передается в службы "активизации" и "чтения".

Менеджер библиотек ISaGRAF автоматически генерирует скелет исходного текста "С" для определения типа структуры данных. Тип структуры данных всегда называется **"str\_data"**. Программист не должен изменять это имя для того, чтобы обеспечить совместимость с заголовками службы. Скрытые данные, обычно, группируют внутренние переменные с образом возвращаемых параметров. Служба "чтения" функционального блока используется для доступа к возвращаемым параметрам и не может быть использована для других операций.

- **Служба инициализации**

Служба инициализации функционального блока вызывается ядром ISaGRAF при запуске приложения. Она позволяет "С" программисту запросить систему разместить память для экземпляра. Ниже - стандартная программа службы инициализации:

```
uint16 FBINIT_xxx (uint16 hinstance)
/* "xxx" is the name of the f. block */
{
    return (sizeof (str_data));
}
```

Аргумент **"hinstance"** - это логический номер экземпляра. Он резервируется ISaGRAF для внутренних операций и не должен использоваться для программирования службы. Служба инициализации возвращает количество байт памяти требуемое для данных одного



экземпляра. Объем требуемой памяти (возвращаемое значение) не может превышать 64К. Никакие другие операции не должны выполняться в этой службе. Исходный текст этой службы автоматически генерируется менеджером библиотек ISaGRAF при создании функционального блока.

- **Служба активизации**

Служба "активизации" вызывается на каждом цикле целевой задачи, для каждого экземпляра функционального блока используемого в приложении. Эта служба обрабатывает параметры вызова и запускает основной алгоритм функционального блока для того, чтобы изменить скрытые статические данные и значения возвращаемых параметров. Ниже - стандартный скелет службы активизации:

```
void FBACT_XXX (
uint16 hinstance,                /* "xxx" is the name of the function block */
                                /* logical number of the instance */
str_data *data,                 /* data: pointer to the instance data structure
*/
str_arg *arg                     /* pointer to the calling parameters structure
*/
)
{
}
```

Аргумент "**hinstance**" - это логический номер экземпляра. Он резервируется ISaGRAF для внутренних операций и не должен использоваться для программирования службы. Аргумент "**data**" - это дальний указатель на структуру данных связанных с экземпляром. Аргумент "**arg**" это дальний указатель на структуру, которая содержит значения параметров вызова. Программист должен использовать идентификаторы, определенные в "C" заголовке функционального блока для доступа к полям структуры "**arg**".

Алгоритм "активизации" обрабатывает параметры вызова (сохраненные в структуре "**arg**") и изменяет поля структуры "**data**".

Следующий пример показывает службу активизации функционального блока **TRIG** (определение переднего фронта):

```
/* definitions stored in the function block "C" header */
```

```
typedef struct {                /* calling parameters */
    T_BOO _clk;                 /* trigger input */
} str_arg;

#define CLK    (arg->_clk)
```

```

/* function block instance data structure */

typedef struct {
    T_BOO prev_state;          /* previous state of the trigger input */
    T_BOO edge_detect;        /* edge value: image of return param */
} str_data;

/* activation service */

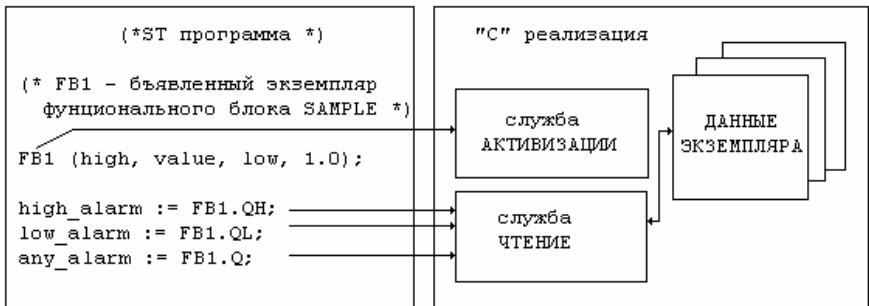
void FBACT_trig (uint16 hinstance, str_data *data, str_arg *arg)
{
    data->edge_detect = (T_BOO)(CLK && !data->prev_state);
    data->prev_state = CLK; /* calling parameter */
}

```

Исходный текст этой службы автоматически генерируется менеджером библиотек ISaGRAF при создании функционального блока.

• **Чтение возвращаемых параметров**

Служба "чтение" вызывается каждый раз, когда ST или FBD программа ссылается на возвращаемый параметр функционального блока. Она используется для того, чтобы получить **один** возвращаемый параметр. Следующий пример показывает вызов "чтения", исполняющийся во время работы ST программы:



Так как служба "чтение" может быть вызвана более одного раза в течение цикла, для одного и того же возвращаемого параметра или экземпляра функционального блока, никаких специальных вычислений не должно выполняться в такой службе. Она только осуществляет передачу между скрытыми данными и приложением ISaGRAF. Ниже - стандартный скелет для службы чтения:

```

/* cast operation used to copy the value of a return parameter */

```

```
#define BOO_VALUE          ((T_BOO *)value)
#define ANA_VALUE          ((T_ANA *)value)
#define REAL_VALUE        ((T_REAL *)value)
#define TMR_VALUE          ((T_TMR *)value)
#define MSG_VALUE          ((T_MSG *)value)
```

/\* return parameters read service: called for each return parameter \*/

```
void FBREAD_XXX (          /* "xxx" is the name of the function block */
uint16 hinstance,         /* logical number of the instance */
str_data *data,          /* pointer to the instance data structure */
uint16 parno,            /* logical number of read parameter */
void *value)             /* buffer where to copy the value of the param
*/
{
    switch (parno) {
        case FBLPNO_XX: /* ... */ break;
        case FBLPNO_YY: /* ... */ break;
        /* .... */
    }
}
```

Аргумент “**hinstance**” - это логический номер экземпляра. Он резервируется ISaGRAF для внутренних операций и не должен использоваться для программирования службы. Аргумент “**data**” - это дальний указатель на структуру данных связанных с экземпляром.

Аргумент “**parno**” - это логический номер возвращаемого параметра, значение которого требуется. Используйте идентификаторы, определенные в “С” заголовке функционального блока для идентификации возвращаемого параметра. Тип данных, на которые указывает этот аргумент, зависит от типа возвращаемого параметра. Ниже представлено соответствие между типами ISaGRAF и “С”:

boolean	long	32 битное беззнаковое слово: 1=true / 0=false
analog	long	32 битное знаковое целое слово
real	float	плавающее значение с одиночной точностью
timer	long	32 битное беззнаковое слово (единица - это 1 миллисекунда)
message	char *	строка символов

Следующие макроопределения используются для доступа к буферу копирования, в соответствии с типом возвращаемого параметра:

```
#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)

```

Вот общие программные операции для копирования значения или параметра к буферу ISaGRAF:

```

/* for a boolean parameter: */
*BOO_VALUE = parameter_value;
/* for an integer analog parameter: */
*ANA_VALUE = parameter_value;
/* for a real integer parameter: */
*REAL_VALUE = parameter_value;
/* for a time parameter: */
*TMR_VALUE = parameter_value;
/* for a string parameter: */
strcpy (*MSG_VALUE, parameter_value);

```

Исходный текст этой службы автоматически генерируется менеджером библиотек ISaGRAF при создании функционального блока.

- **Пример исходного текста**

Ниже представлен стандартный скелет “С” функционального блока:

```

/* function block (xxx is the name of the function block) */

#include <tasy0def.h>
#include <grfb0nnn.h> /* nnn is the number of the f.block in library */

/* structure of hidden data for each instance of the block */
typedef struct {
    /* fields definition */
} str_data;

/* initialization service: returns the size of needed hidden data */
word FBINIT_xxx (uint16 hinstance)
{
    return (sizeof (str_data));
}

/* activation service: processes the calling parameters */
void FBACT_xxx (uint16 hinstance, str_data *data, str_arg *arg)

```

```

{
    /* ... */
}

/* cast operation used to copy the value of a return parameter */
#define BOO_VALUE ((T_BOO *)value)
#define ANA_VALUE ((T_ANA *)value)
#define REAL_VALUE ((T_REAL *)value)
#define TMR_VALUE ((T_TMR *)value)
#define MSG_VALUE ((T_MSG *)value)

/* return parameters read service: called for each return parameter */
void FBREAD_XXX (uint16 hinstance, str_data *data, uint16 parno, void *value)
{
    switch(parno)
    {
        case FBLPNO_XX: *???_VALUE = ...; break;
        case FBLPNO_YY: *???_VALUE = ...; break;
        ....
    }
}

/*Следующая функция используется для инициализации функционального
блока и объявления его реализации. Она реализует связь с ядром ISaGRAF,
используя имя функции. Эта функция полностью генерируется менеджером
библиотек ISaGRAF*/

ABP fbldef_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 */

```

Включенный файл **“TASY0DEF.h”** из ядра ISaGRAF требуется для системозависимых определений. Он, также содержит определения типов данных, которые представляют дальние указатели на реализованные службы.

## ▬ Связь между проектами и “С” реализацией

Логическая связь между реализацией функции и ее использованием в программах проекта ISaGRAF осуществляется через имя функции. Служба объявления добавляется в исходный “С” код функционального блока. Эта служба вызывается только один раз, когда приложение стартует и сообщает ядру ISaGRAF имя, которое соответствует реализованной службе. Вот стандартный формат такой службы объявления:

```
ABP fbldef_xxx (char *name, IBP *initproc, RBP *readproc)
{
    strcpy (name, "XXX");           /* name of the f.block */
    *initproc = (IBP)FBINIT_xxx;   /* initialization service */
    *readproc = (RBP)FBREAD_xxx;   /* read service */
    return ((ABP)FBACT_xxx);       /* activation service */
}
/* xxx is the name of the function block */
```

Имя функционального блока, использующееся оператором **strcpy**, должно быть написано заглавными буквами. Оно должно быть написано маленькими буквами в реализации служб и в имени службы объявления.

Использование префиксов “FBACT\_”, “FBINIT\_”, “FBREAD\_” и “fbldef\_” для реализации служб и определения службы позволяет пользователю именовать функции ключевыми словами языка “С” или именами существующих функций из “С” библиотек ISaGRAF. Никакие другие операторы нельзя добавлять в службу объявления.

Служба объявления вызывается для любого интегрированного функционального блока, даже если он не используется в приложении ISaGRAF. Ядро ISaGRAF выдает ошибку, если в приложении используется не реализованный функциональный блок.

Прежде чем линковать новые функциональные блоки с ядром, пользователь должен написать другой файл с исходным текстом, под именем “GRFB0LIB.C” и вставить его с сохраненным функциональным блоком в список файлов для линкера. “GRFB0LIB.C” содержит только массив служб объявления. Этот массив читается во время инициализации приложения, для того чтобы создать динамические связи с функциональными блоками написанными на “С”. Вот пример такого файла:

```
/* Файл: grfb0lib.c - implemented function blocks */

#include <tasy0def.h>

extern ABP fbldef_fb1(char *name, IBP *init, RBP *read);
extern ABP fbldef_fb2(char *name, IBP *init, RBP *read);

FBL_LIST FBLDEF[ ] = {
    fbldef_fb1,
```

```

    fbldf_fb2,

NULL };

/* конец файла */

```

Массив **FBLDEF** должен заканчиваться указателем NULL. Если это условие не выполнено, то могут возникнуть некоторые проблемы. Если массив **FBLDEF** не определен, то во время линковки нового ядра ISaGRAF появятся неразрешенные ссылки.

Написав этот файл, можно построить новое ядро, включая все существующие функции. Можно, также построить ядро, приспособленное для одного проекта, путем введения в массив **FBLDEF** только тех преобразований, которые используются в проекте. Файл **“GRFB0LIB.C”** автоматически генерируется генератором кодов ISaGRAF, когда создается код приложения. Файл помещается в директорию проекта ISaGRAF и объединяет только те функции, которые используются в проекте.

### ⇒ **Ограничения**

Библиотека ISaGRAF может содержать до 255 “С” функциональных блоков. Функция может выполнять любой тип операции. Каждый тип функционального блока может быть скопирован (сделано экземпляров) до 255 раз в одном проекте.

Необходимо заметить, что функциональные блоки вызываются в цикле ISaGRAF **синхронно**, так что выполнение функционального блока оказывает непосредственное влияние на временной цикл.

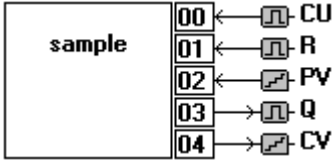
### ⇒ **Полный пример**

Ниже представлен пример функционального блока “sample”, который выполняет счет вверх.

Техническое замечание:

имя:	Sample
описание:	счетчик вверх
дата:	18 мая 1995
автор:	CJ International
вызов:	CU: считающийся вход R: команда сброса PV: максимальная планируемая величина
возврат:	Q: определение максимума CV: результат счета
прототип:	sample (count, reset_command, maximum_value); max_detect := SAMPLE.Q; count_result := SAMPLE.CV;

Интерфейс функционального блока:



Заголовок функционального блока:

```

/* function block interface - name: SAMPLE */

/* определение стандартных типов ISaGRAF */

typedef long T_BOO;
typedef long T_ANA;
typedef float T_REAL;
typedef long T_TMR;
typedef char *T_MSG;

/* определение структуры параметров вызова */

typedef struct {
    T_BOO _cu;
    T_BOO _r;
    T_ANA _pv;
} str_arg;

/* идентификаторы использующиеся для доступа к параметрам вызова*/

#define CU (arg->_cu)
#define R (arg->_r)
#define PV (arg->_pv)

/* логические номера возвращаемых параметров */

#define FBLPNO_Q 0
#define FBLPNO_CV 1

/* end of file */

```

Исходный текст функционального блока:

(Только строки напечатанные жирным шрифтом вводятся программистом)



```

/* function block - name: SAMPLE */

#include <tasy0def.h>           /* требуется для определения типов */
#include <grfb0255.h>         /* заголовок*/

/* definition of the structure which contains the data for one instance */

typedef struct {
    T_BOO overflow;        /* true:counting value >= programmed value */
    T_ANA value;          /* counting current value */
} str_data;

/* initialization service: requires memory for instance data */

word FBINIT_sample (uint16 hinstance)
{
    return (sizeof (str_data));
}

/* activation service: up-counting 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;
}

/* cast operation required to copy parameters to ISaGRAF buffer */

#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)

/* read service: get the value of one return parameter */

```

```

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;
    }
}

/* declaration service used for dynamic link with the ISaGRAF kernel */

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 */

```

### С.7.5 Техника компилирования и линкования.

Система разработки ISaGRAF не включает в себя “С” компиляторов или линкеров. Однако эта глава объясняет основную технику, которая может быть применена для легкого использования файлов, созданных Менеджером Библиотек ISaGRAF, и затем используемые другим инструментарием, таким как компиляторы и линковщики.

#### ▣ **Исходные “С” файлы**

Преобразуемые исходные “С” файлы, функции и функциональные блоки размещены Менеджером Библиотек ISaGRAF в директории ISAWIN\LIB\DEFS и ISAWIN\LIB\SRC. Имя исходного файла содержит номер соответствующего преобразования, функции или функционального блока в библиотеке ISaGRAF. Используемые имена файлов:

\isawin\lib\defs\TACN0DEF.H	файл определений для любых функций преобразования
\isawin\lib\src\GRCN0nnn.H	исходный файл функций преобразования
\isawin\lib\defs\GRUS0nnn.H	файл определений функций
\isawin\lib\src\GRUS0nnn.C	исходный файл функций
\isawin\lib\defs\GRFB0nnn.H	файл определений функциональных блоков

`\isawin\lib\src\GRFB0nnn.C` исходный файл функциональных блоков

(nnn - номер преобразования, функции или функционального блока)

**Предупреждение:** Когда переименовываются или копируются элементы библиотеки, ее текстовые и программные строки не обновляются Менеджером Библиотек ISaGRAF в соответствии с новыми именами элементов и логическими номерами. Они должны быть обновлены вручную в исходном "C" файле.

Файл **ISAWINLIBUSPNUMS** дает соотношение между именами и логическими номерами для "C" функций, расположенных в библиотеке ISaGRAF. Пример такого файла:

```
1    funct_A
10   funct_B
16   funct_C
```

Файл **ISAWINLIBFBLNUMS** дает соотношение между именами и логическими номерами для "C" функциональных блоков, расположенных в библиотеке ISaGRAF.

Пример такого файла:

```
0    fbl_A
1    fbl_B
2    fbl_C
```

Файл **ISAWINLIBCNVNUMS** дает соотношение между именами и логическими номерами для функций преобразования, расположенных в библиотеке ISaGRAF.

Пример содержимого файла для стандартной библиотеки преобразований:

```
0    SCALE
1    BCD
```

Эти файлы автоматически обновляются Менеджером Библиотек ISaGRAF каждый раз при преобразовании, создании функции или функционального блока, переименовании, копировании или уничтожении функционального блока. Генератор кода ISaGRAF генерирует следующие файлы при создании приложения:

<code>\isawin\apl\ppp\GRCN0LIB.C</code>	Объявление в виде массива всех функций, преобразования используемых в проекте.
<code>\isawin\apl\ppp\GRUS0LIB.C</code>	Объявление в виде массива всех функций, используемых в проекте.
<code>\isawin\apl\ppp\GRFB0LIB.C</code>	Объявление в виде массива всех функциональных блоков, используемых в проекте.

(ppp имя проекта ISaGRAF)

Эти файлы могут использоваться во время выполнения операций линкования для построения ядра ISaGRAF, предназначенного для проекта, который содержит преобразования, функции и функциональные блоки, используемые только в этом проекте.

## ≡ **Загрузка исходных файлов в систему**

Исходные “С” файлы и файлы определений, созданные Менеджером Библиотек ISaGRAF, могут быть загружены в целевую задачу системы ISaGRAF, если это поддерживается родным компилятором. Для этого может быть использован стандартный инструмент **TERMINAL**, поставляемый с Windows.

Когда исходные файлы управляются в целевой системе, файлы определений будут обновляться новыми операциями загрузки каждый раз, когда функциональный интерфейс изменяется Менеджером Библиотек ISaGRAF.

Командные строки для загрузки файлов могут быть сгруппированы, например, в командных файлах, а затем запущены из меню инструментов системы разработки (смотри руководство пользователя: Управление программами)

## ≡ **Использование кросс - компилятора**

С исходными файлами можно работать непосредственно на вашем PC, если PC является целевой системой, или доступен кросс - компилятор, запущен на PC и генерирует коды для целевой системы.

В этом случае, пользователь может запустить Менеджер Библиотек ISaGRAF для создания и модификации исходных текстов преобразований, функций и функциональных блоков. Командные строки для загрузки файлов могут быть сгруппированы, например в командных файлах, а затем запущены из меню инструментов системы разработки (смотри руководство пользователя: Управление программами)

Когда преобразования, функции и функциональные блоки скомпилированы на PC, пользователь просто загружает вновь сгенерированное ядро ISaGRAF (линкованное с новыми компонентами) в целевую задачу системы перед запуском приложений. Если целевой системой является другой PC, то вновь сгенерированное ядро ISaGRAF может быть загружено в эту машину с дискеты или через сеть.

## ≡ **Линкование с библиотеками ядра ISaGRAF**

### **Предупреждение:**

Далее следует общая информация, которая может не совсем соответствовать вашей целевой системе.

В любом случае вы можете справиться в readme и .TXT файлах, поставляемых на целевом диске.

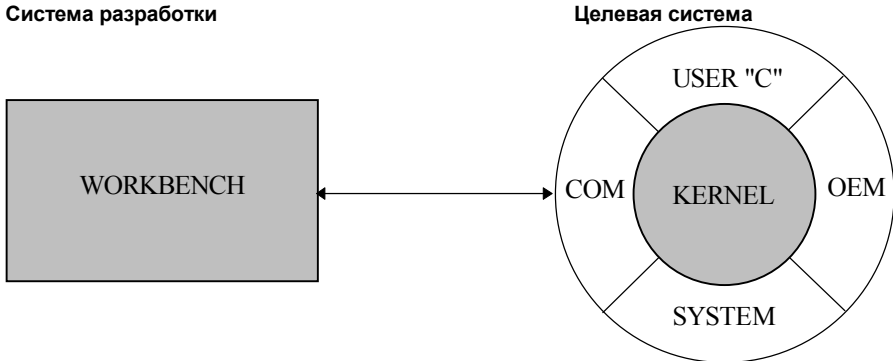
Целевая дискета ISaGRAF содержит много файлов утилит для компиляции и линкования преобразований, функций и функциональных блоков библиотеками ядра ISaGRAF.

Существует две реализации:

- однозадачный ISaGRAF: все функции выполняются в одной программе
- многозадачный ISaGRAF: для связи выделяется отдельная задача (или нить)

В любом случае "С" компоненты сгруппированы в одинаковых библиотеках: для "С" программистов нет разницы в создании однозадачной или многозадачной системы. Для однозадачной версии "С" библиотеки пользователя линкуются в одну задачу (обычно называемую isa), тогда как для многозадачной версии библиотеки линкуются в задачу ядра (обычно называемую isaker).

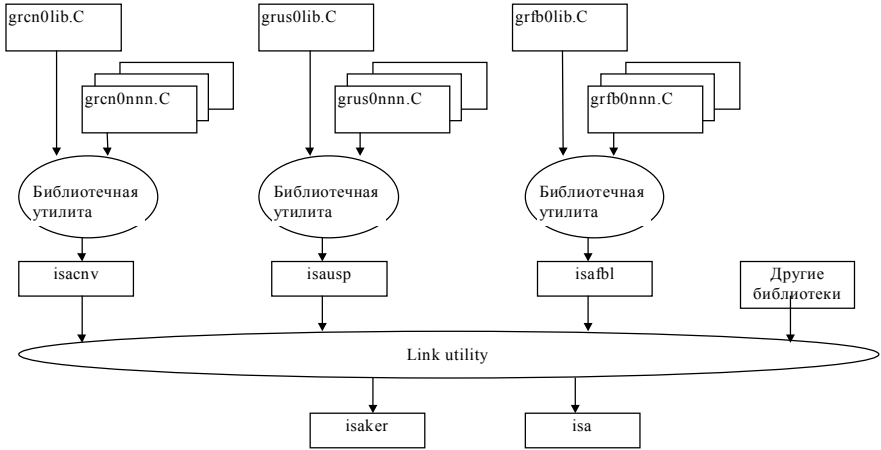
**Система разработки**



Внутренняя часть программного обеспечения ISaGRAF не зависит от аппаратной части. Она выполняет IEC языки и имеет свою собственную базу данных переменных. Первым шагом создания связи с ядром является построение библиотек преобразований, функций и функциональных блоков, необходимых для конкретного проекта:

Библиотеки	Содержание
ISAUSP	- объектный файл GRUSOLIB (массив декларированных функций) - объектный файл каждой интегрированной функции
ISAFBL	- объектный файл GRFBOLIB (массив декларированных функциональных блоков) - объектный файл каждого интегрированного функционального блока
ISACNV	- объектный файл GRCNOLIB (массив декларированных преобразований) - объектный файл каждой интегрированной функции преобразования

Затем программист должен слинковать эти новые библиотеки с другими объектными файлами и библиотеками ядра ISaGRAF. Различные стадии разработки "С" приложений показаны на следующей диаграмме:



Список объектных модулей и библиотек, которые присоединяются во время линкования:

Построение isaker:

Объектный Модуль: **tast0mal**  
 Объектный Модуль: **tast0com**

Библиотека ядра: **isaker**  
 Библиотека ядра: **isaoem**

Библиотека пользователя: **isausp**                   определенные пользователем функции

Библиотека пользователя: **isafbl**                   определенные пользователем функциональные блоки

Библиотека пользователя: **isacnv**                   определенные пользователем функции преобразования

Библиотека ядра: **isasys**

Системные библиотеки: (refer to your "C" compiler manual)

Построение isa

Объектный Модуль: **tast0mal**  
 Объектный Модуль: **tast0com**

Библиотека ядра: **isaker**  
 Библиотека ядра: **isatst**  
 Библиотека ядра: **isaoem**

Библиотека пользователя: **isausp**                   определенные пользователем

Библиотека пользователя:	<b>isafbl</b>	функции определенные пользователем
Библиотека пользователя:	<b>isacnv</b>	функциональные блоки определенные пользователем
Библиотека ядра:	<b>isasys</b>	функции преобразования
Системные библиотеки:	(refer to your "C" compiler manual)	

Программист может придерживаться порядка следования объектных модулей и библиотек, показанного на предыдущем рисунке. Объектные модули и библиотеки имеют стандартные расширения (".lib", ".obj", ".l", ".r" ...) в соответствии с целевой системой.

### ▬ **Требующиеся опции компиляции**

Опции преобразования могут быть выбраны во время компилирования и линкования. Они зависят от типа операций, выполняемых в преобразованиях, функциях и функциональных блоках. Некоторые операции требуют другие системные библиотеки (math, graphics ...) во время линкования.

Все исходные файлы "C" Ядра ISaGRAF были компилированы в **LARGE** модели памяти. Программист должен использовать одинаковые модели для компилирования преобразований, функций и функциональных блоков.

При компилировании компонентов "C" библиотеки определяется специальная константа. Она указывает тип целевой системы и процессоров, для того, чтобы исходные тексты преобразований, функций и функциональных блоков были независимыми от системы. Ниже указаны значения этой константы:

- DOS**.....для DOS базированных систем (процессор NTEL)
- ISAWNT**.....для Windows-NT базированных систем (процессор INTEL)
- OS9**.....для систем OS9 (процессор MOTOROLA)
- VxWorks**.....для систем VxWorks (процессор MOTOROLA)

Файлы командных утилит (для компиляции и линкования), поставляемые с целевым программным обеспечением ISaGRAF, показывают как определить соответствующее значение константы для командной строки компилятора.

### ▬ **Поддерживаемые компиляторы**

Поддерживаются следующие компиляторы для создания преобразований, функций и функциональных блоков, и линковки с ядром ISaGRAF:

Компилятор Microsoft MSC 7.00	для целевой задачи DOS
Компилятор Microsoft MSVC 4.00	для целевой задачи Windows-NT
Компилятор Microware ULTRA-C	для целевой задачи OS-9
Tornado 1.0; GNU Toolkit 2.6	для целевой задачи VxWorks

По поводу использования других компиляторов контактируйте с CJ International.

## **☰ Резюме**

Ниже приведена краткая сводка операторов, которые будут выполняться во время создания новых преобразований, функций или функциональных блоков:

- ☐1. Используется с Менеджером Библиотек ISaGRAF для создания новых элементов: дает им имена и текстовые комментарии. Автоматически генерируется фрейм "С" исходного файла.
- ☐2. Используется с Менеджером Библиотек ISaGRAF, описывает интерфейс (вызывает и возвращает параметры) если элементы являются функциями или функциональными блоками. Автоматически генерируются заголовки исходных "С" файлов.
- ☐3. Используется с Менеджером Библиотек ISaGRAF, вводит текст подробного технического описания элемента (руководство пользователя).
- ☐4. Используется с Менеджером Библиотек ISaGRAF, завершает исходный файл "С" путем добавления "С" программирования преобразования, функции или алгоритма функционального блока. Исходный код элемента теперь завершен. Можно использовать другой редактор.
- ☐5. Выбор опции "Показать логический номер" Менеджера Библиотек, чтобы узнать, какой логический номер присоединен к новому элементу. Этот номер используется в имени пути преобразования ".С" и ".Н" исходных файлов.
- ☐6. Копировать/Загружать .С и .Н файлы в вашу целевую систему (если родной компьютер) или в соответствующее окружение (если кросс компьютер), где находятся целевые библиотеки ISaGRAF и инсталлированы задачи.
- ☐7. Запустить компилятор "С" на новом исходном файле и корректировать имеющиеся синтаксические ошибки.
- ☐8. Вставить имя нового элемента определенного сервиса в "GR??0LIB.C" исходный файл, который определяет массив вставленных элементов.
- ☐9. Запустить компилятор "С" для компилирования файла "GR??0LIB.C"..
- ☐10. Вставить имя объектного модуля в список объектных файлов, используемых для построения соответствующих библиотек..
- ☐11. Запустить построитель "С" библиотек. Запустить линковщик "С" для построения нового ядра.
- ☐12. Установить вновь созданное ядро на вашей целевой машине.
- ☐13. Запустить пример приложения ISaGRAF, которое тестирует вызов и интерфейс нового элемента.



## С.8 Связь по Modbus

Если приложение полностью разработано и протестировано, вы можете присоединить его к системе визуализации процесса.

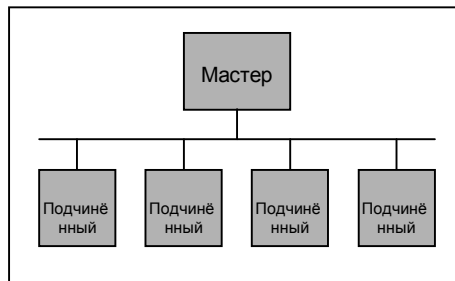
ISaGRAF является открытой системой, предлагающей большое число разнообразных сетевых возможностей.

Простейшей промышленной сетью является протокол в стандарте MODBUS/MODICON, который доступен почти на каждой системе визуализации процесса, и который требует только последовательную связь (RS232, RS485, Токовая петля).

Отладочный протокол связи ISaGRAF является MODBUS совместимым для использования различных видов доступа чтение/запись из Modbus мастера.

### С.8.1 Сеть MODBUS и протокол

Сеть Modbus состоит только из одной мастер станции (обычно система визуализации процесса) и одной или более подчиненных станций (обычно PLC).



Мастер каждый раз посылает один запрос одному подчиненному (используя номер подчиненного) и ждет ответа от подчиненного прежде чем послать второй запрос. Другой свободный подчиненный не отвечает.

Каждый фрейм содержит номер подчиненного, номер запроса и соответствующие данные, а также 16-и битный код контрольной суммы(CRC).

Если за время тайм-аута ответ не приходит, запрос может быть повторен некоторое число раз до того, как мастер объявит подчиненный "отсоединенным".

Значение тайм-аута и число повторений устанавливается на мастер станции для установки запросов к подчиненному (в зависимости от приложения, и т.д.).

Если при обработке запроса обнаруживается ошибка, то подчиненный может послать сообщение об ошибке вместо ожидаемого ответного фрейма.

Modbus является протоколом Modicon, а не международным стандартом, который имеет много различных реализаций Modbus совместимых протоколов, с большим числом возможных вариантов, например:

- Список поддерживаемых функциональных кодов
- Отображение адресов
- RTU (двоичный код) или ASCII протокол и т.д.

## С.8.2 Реализация ISaGRAF

### ⇒ **Доступ к прикладным переменным**

Коммуникационное соединение ISaGRAF распознает пять функциональных кодов Modbus:

1	читать N бит
3	читать N слов
5	писать 1 бит
6	писать 1 слово
16	писать N слов

Доступ к переменным приложения ISaGRAF может осуществляться через их "сетевой адрес", если конечно они были определены в библиотеке инструментальных средств.

Эти переменные могут быть:

- Булевыми или Аналоговыми переменными
- Входными, выходными или терминальными переменными
- Локальными или глобальными переменными.

Для записи Булевой переменной могут использоваться функции 5, 6 или 16. Значение TRUE для записи является любым ненулевым значением.

Для чтения Булевой переменной могут использоваться функции 1 или 3. Функция 1 возвращает значение в битовое поле, а функция 3 - в байтовое (значение TRUE соответствует значению 0xFFFF).

Для записи Аналоговой переменной могут использованы функции 6 и 16. Значение является 16-ти битовым целым в диапазоне от -32768 до +32768 (целые переменные ISaGRAF являются 32-х битовыми).

Для чтения Аналоговой переменной может использоваться функция 3. .  
Переменные типа Real не доступны по запросам Modbus.

#### Предупреждение:

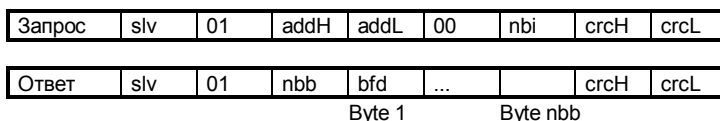
Реализация ISaGRAF не управляет такими кодами ошибки как 'неизвестный адрес modbus'.

#### **Обозначения:**

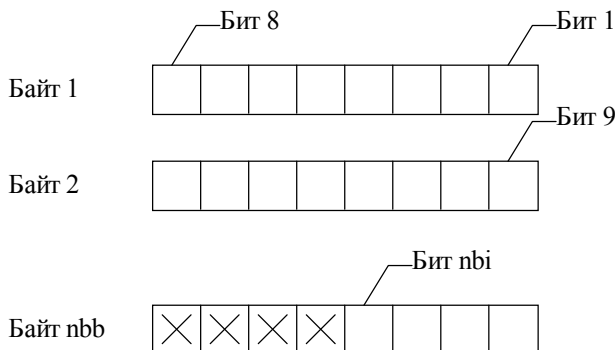
slv	номер подчиненного
nbw	число слов
nbb	число байт
nbi	число бит
addH	Сетевой адрес (старший байт)
addL	сетевой адрес (младший байт)
vH	значение (старший байт)
vL	значение (младший байт)
V	байтовое значение
bfd	битовое поле (nbb байт)
crсH	контрольная сумма (старший байт)
crсL	контрольная сумма (младший байт)

### Функция 1: читать n бит

Читает nbi бит (булевских значений), начиная с сетевого адреса addH/addL



где bfd является битовым полем длиной nbb байт следующего формата



Бит 1 соответствует значению переменной с сетевым адресом addH/addL.  
 Бит nbi соответствует значению переменной с сетевым адресом addH/addL+nbi-1.  
 X означает, что значение не используется

### Функция 3: читать n слов

Читает nbw слов, начиная с сетевого адреса addH/addL

Запрос	slv	03	addH	addL	00	nbw	crcH	crcL
--------	-----	----	------	------	----	-----	------	------

Ответ	slv	03	nbb	vH	vL	...	crcH	crcL
-------	-----	----	-----	----	----	-----	------	------

где nbb соответствует числу байт во всех парах vH,vL.

**Функция 5: записать 1 бит**

Записать бит (булевскую переменную) по сетевому адресу addH/addL

Запрос	slv	05	addH	addL	vH	00	crcH	crcL
--------	-----	----	------	------	----	----	------	------

Ответ	slv	05	addH	addL	vH	00	crcH	crcL
-------	-----	----	------	------	----	----	------	------

**Функция 6: записать 1 слово**

Записать слово по сетевому адресу addH/addL

Запрос	slv	06	addH	addL	vH	vL	crcH	crcL
--------	-----	----	------	------	----	----	------	------

Ответ	slv	06	addH	addL	vH	vL	crcH	crcL
-------	-----	----	------	------	----	----	------	------

**Функция 16: записать N слов**

Записать nbw слов, начиная с сетевого адреса addH/addL (nbb=2nbw)

Запрос	slv	10	addH	addL	00	nbw	nbb	vH	vL	...	crcH	crcL
--------	-----	----	------	------	----	-----	-----	----	----	-----	------	------

Ответ	slv	10	addH	addL	00	nbw	crcH	crcL
-------	-----	----	------	------	----	-----	------	------

**Примеры:**

– Функция 1: прочитать 15 бит, начиная с сетевого адреса 0x1020 на подчиненном 1

Запрос	01	01	10	20	00	0F	79	04
--------	----	----	----	----	----	----	----	----

Ответ	01	01	02	00	12	39	F1
-------	----	----	----	----	----	----	----

считанное значение - 0x0012, что в битовом представлении эквивалентно 00000000 000100101.

В этом примере переменные 0x1029 и 0x102C имеют значение TRUE, остальные - FALSE.

– Функция 16: записывает 2 слова по адресу 0x2100 на slave 1, записанные значения равняются: 0x1234 и 0x5678.

Запрос	01	10	21	00	00	02	04	12	34	56	78	1C	CA
--------	----	----	----	----	----	----	----	----	----	----	----	----	----

Ответ	01	10	21	00	00	02	4B	F4
-------	----	----	----	----	----	----	----	----

## Перенос файлов

По сравнению с современными промышленными шинами протокол Modbus обеспечивает очень скромные возможности, если только набор функций не расширен поставщиком.

В нашей ситуации при запуске ISaGRAF имеются два ограничения на протокол Modbus

- Доступны только переменные ISaGRAF
- Невозможна быстрая передача больших объемов данных

Имеются причины, по которым ISaGRAF поддерживает набор 'Modbus-подобных' запросов для переноса файлов или протокол 'управления удаленными файлами':

- Дистанционная загрузка ASCII и бинарных файлов
- Подгрузка ASCII и бинарных файлов
- Динамический обмен данными через виртуальный или физический разделяемый файл

Таким образом, при наличии коммуникационной связи с ISaGRAF любое приложение, независимое от ISaGRAF, может легко связываться с удаленной целевой задачей.

Протокол базируется на следующих понятиях:

- Файл со стороны целевой задачи ISaGRAF называется **удаленным файлом**
- Файл на мастер-компьютере называется **локальным файлом**
- Каждый байт в файле имеет 32-битовый **базовый адрес** и 16-битовый **адрес байта**

Имеются запросы выбора имени удаленного файла, выбора базового адреса, считать или записать данные в удаленный файл, используя 16-битовый адрес байта.

### Функция 17: записать байты

nbb соответствует числу байт во всех парах vH, vL

Запрос	slv	11	addH	addL	00	nbb	nbb	vH	vL	...	crch	crcl
--------	-----	----	------	------	----	-----	-----	----	----	-----	------	------

Ответ	slv	11	addH	addL	00	nbb	crch	crcl
-------	-----	----	------	------	----	-----	------	------

Значение этого запроса различается в зависимости от значения адреса addH/addL:

- **0xF000: Инициализировать имя удаленного файла**  
nbb соответствует числу символов в имени файла, располагающегося в полях vH, vL и т.д. (в этом случае понятия 'старший' и 'младший' лишены смысла), включая символ \0 окончания строки. Если файл не существует, то он создается с атрибутами writable+readable+executable.
- **0xF002: Изменить базовый адрес на указанное значение**  
nbb должно равняться 4. Первая пара vH, vL есть значение старшего слова значения адреса. Возможно любое 32-битное значение. Все последующие запросы чтения и

записи будут использовать этот базовый адрес. Если запроса изменения базового адреса не делалось, по умолчанию будет использоваться нулевое значение.

– **0xF004: Уничтожить файл**

nbb должно равняться 0. Файл будет уничтожен если он существует и если это возможно.

– **Больше 0xF004: Зарезервировано**

– **Меньше 0xF000: Записать байты**

Адрес байта, по которому нужно произвести запись указывается в addH/addL. Он должен быть меньше 0xF000. Записываются значения nbb байт, указываемые в полях vH,vL и т.д., в заданном порядке (слева - направо) в удаленный файл, имя которого ранее было выбрано. Запись производится по адресу, который получается сложением ранее установленного базового адреса с указываемым адресом байта. Если результирующий адрес выходит за границу файла, размер файла увеличивается. Невозможно уменьшить размер файла.

**Функция 18: читать данные**

Запрос	slv	12	addH	addL	00	nbb	crcH	crcL
--------	-----	----	------	------	----	-----	------	------

Ответ	slv	12	nbb	V	V	...	crcH	crcL
-------	-----	----	-----	---	---	-----	------	------

Адрес для чтения указывается в addH/addL. Он должен быть меньше F000. Происходит считывание nbb байт из удаленного файла, чье имя ранее было выбрано, начиная с адреса, получаемого добавлением адреса байта addH/addL к ранее установленному базовому адресу. Значения переносятся в порядке как они лежат в файле (слева - направо).

**Пример:**

Выбрать имя удаленного файла: target.fil.

Запрос	01	11	F0	00	00	0B	0B	74	...	00	25	9F
--------	----	----	----	----	----	----	----	----	-----	----	----	----

Ответ	01	11	F0	00	00	0B	8F	0E
-------	----	----	----	----	----	----	----	----

Выбрать базовый адрес 0x10000.

Запрос	01	11	F0	02	00	04	04	00	01	00	00	76	11
--------	----	----	----	----	----	----	----	----	----	----	----	----	----

Ответ	01	11	F0	02	00	04	6E	CA
-------	----	----	----	----	----	----	----	----

Записать 4 байта: абсолютный адрес - 0x107D0, значения - 01, 02, 03, 04.

Запрос	01	11	07	D0	00	04	04	01	02	03	04	28	6F
--------	----	----	----	----	----	----	----	----	----	----	----	----	----

Ответ	01	11	07	D0	00	04	FC	87
-------	----	----	----	----	----	----	----	----

Считать 4 байта: абсолютный адрес - 0x107D0.

Запрос	01	12	07	D0	00	04	B8	87
--------	----	----	----	----	----	----	----	----

Ответ	01	12	04	01	02	03	04	58	7D
-------	----	----	----	----	----	----	----	----	----

## С.9 Управление при отказе питания

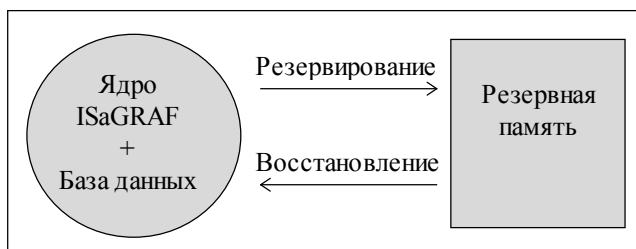
### С.9.1 Основы

Управление при отказе питания имеет большое значение для приложения по следующим причинам:

- Оно зависит от особенностей процесса
- Оно зависит от возможностей аппаратуры
- Оно зависит от методов программирования

Поэтому решение, заложенное в ISaGRAF не претендует на полноту, универсальность и абсолютность, а представляет собой набор принципов, методов и инструментальных средств, которые должны комбинироваться с учетом особенностей конкретного приложения или, по меньшей мере, аппаратуры.

Для того, чтобы успешно перезапустить процесс в системе управления, после наступления отказа питания, должны быть решены 3 задачи:



- Создание резервной копии
- При старте системы распознавание ситуации, что имел место отказ питания
- Восстановление сохраненных данных

Вторая проблема не имеет стандартного решения одними лишь программными средствами, но поставщик может обеспечить соответствующие средства доступа для определения состояния аппаратуры из приложения ISaGRAF или программы С.

Далее, важно решить какие данные следует сохранять и восстанавливать. Определяется два вида данных:

– Переменные приложения:

Такие переменные процесса как число обрабатываемых элементов, данные об отказе питания, значения параметров процесса и т.д.

Такие программные переменные, как счетчики, таймеры, промежуточные значения и флаги.



- Состояние программы:  
Это указатель активных шагов, статус каждой “С” программы и т.д.

Эти два случая изучаются в следующих главах.

## С.9.2 Сохранение переменных приложения

### ⇒ **Сохраняемые переменные**

Редактор переменных системы разработки поддерживает атрибут ‘сохранять’ для каждой внутренней переменной (т.е. не входной и не выходной).

В конце каждого цикла целевой задачи значения сохраняемых переменных копируются в специальную область памяти. Этой областью обычно является питаемая от батареи RAM.

Во время старта если хотя бы одна переменная имела атрибут сохраняемой, ISaGRAF ищет сохраняемые переменные:

- Если данное приложение уже запускалось ранее, ISaGRAF находит сохраненные значения и присваивает их соответствующим сохраняемым переменным.
- Если ранее работало другое приложение или не было никакого, то ISaGRAF понимает, что сохраненные значения недействительны и переустанавливает все сохраняемые переменные в 0.

Область памяти для сохранения переменных определяется в системе разработки в меню **Make/Опции выполнения приложения/Сохраняемые переменные**. Указанная строка должна иметь следующий формат:

<code>boo_add , boo_size , ana_add , ana_size , tmr_add , tmr_size , msg_add , msg_size</code>
--

где:

- boo\_add** Шестнадцатеричный адрес, используемый для сохранения булевских переменных. Должен быть отличен от 0.
- boo\_size** Шестнадцатеричный размер памяти в байтах, выделяемый по этому адресу. Для хранения булевской переменной требуется 1 байт.
- ana\_add** Шестнадцатеричный адрес, используемый для сохранения аналоговых переменных. Должен быть отличен от 0.
- ana\_size** Шестнадцатеричный размер памяти в байтах, выделяемый по этому адресу. Всегда требуется минимум 4 байта плюс 4 байта для хранения каждой аналоговой переменной.
- tmr\_add** Шестнадцатеричный адрес, используемый для сохранения таймерных переменных. Должен быть отличен от 0.
- tmr\_size** Шестнадцатеричный размер памяти в байтах, выделяемый по этому адресу. Для хранения таймерной переменной требуется 5 байт.

- msg\_add** Шестнадцатеричный адрес, используемый для сохранения строковых переменных. Должен быть отличен от 0.
- msg\_size** Шестнадцатеричный размер памяти в байтах, выделяемый по этому адресу. Для хранения таймерной переменной требуется 256 байт.

### Требования

- Должны быть специфицированы все поля для всех типов даже если Вы не собираетесь сохранять все типы переменных. В таком случае для неиспользуемого типа переменных Вы должны указать нулевой размер (за исключением аналогов, где минимальный размер - 4) и любой отличный от нуля адрес.

### Пример:

Предположим, требуется резервирование

- 20 булевских переменных
- 0 аналоговых переменных
- 0 таймерных переменных
- 3 строковых переменных

Пусть память, питаемая от батареи, находится по адресу 0xA2F200.

Предположим, что:

Булевские переменные будут храниться, по адресу 0xA2F200 и будут занимать в точности 20 байт.

Аналоги будут по адресу 0xA2F214, и для них нужно 4 байта.

Фиктивный адрес для таймеров будет 0xA2F200, т.к. размер памяти - нулевой.

Строковые переменные будут храниться по адресу 0xA2F218, причем им требуется 3\*256 байт.

Таким образом, в системе разработке должна быть введена строка

A2F200,14,A2F214,4,A2F200,0,A2F218,300
--

## ≡ **Функциональный вызов SYSTEM**

Если нужно хранить большинство переменных приложения, то следует использовать функцию SYSTEM, чтобы иметь дело со всем множеством переменных (подробнее о функции SYSTEM см. в Руководстве пользователя). Заметим, что в этом случае резервирование и восстановление управляются программистом на уровне приложения.

Прежде всего, Вы должны определить место расположения резервных копий всех типов переменных:

**<новый\_адрес>:=SYSTEM(SYS\_INITxxx,<адрес>);**

где

- <адрес> адрес расположения памяти для резервирования (16#шестнадцатеричное значение). Он должен иметь четное значение, в противном случае операция не выполнится.
- SYS\_INITxxx может быть:
  - \* SYS\_INTBOO для определения начала памяти под булевские переменные.
  - \* SYS\_INTANA для определения начала памяти под аналоговые переменные.
  - \* SYS\_INTTMR для определения начала памяти под таймерные переменные.
  - \* SYS\_INTALL для определения начала памяти под все булевские, аналоговые и таймерные переменные.

- <новый\_адрес> значение следующего свободного адреса, который равен <адрес> + зарезервированная память в байтах согласно SYS\_INITxxx. Это позволяет проконтролировать выделенный объем памяти. Если операция не выполнялась, <новый\_адрес> получает нулевое значение.

Теперь можно сделать запрос на резервирование. Эту процедуру можно вызвать в любой момент в приложении; резервирование будет выполнено в конце текущего цикла. Если аппаратура дает возможность информировать пользователя с помощью булевого входа или С функции об отказе питания и обеспечивает хотя бы один цикл ISaGRAF после этого, резервирование можно делать только при обнаружении такого отказа:

**<ошибка>:=SYSTEM(SYS\_SAVxxx,0);**

где

- SYS\_SAVxxx может быть:
  - \* SYS\_SAVBOO для резервирования всех булевских переменных.
  - \* SYS\_SAVANA для резервирования всех аналоговых переменных.
  - \* SYS\_SAVTMR для резервирования всех таймерных переменных.
  - \* SYS\_SAVALL для резервирования всех булевских, аналоговых и таймерных переменных.
- <ошибка> получает значение ненулевого кода ошибки если операция не выполнялась. (SYS\_INITxxx не отработала).

Наконец, Вы можете пожелать восстановить значения зарезервированных переменных. Эту процедуру можно вызвать в любой момент в приложении; резервирование будет выполнено в конце текущего цикла. Формат команды восстановления следующий

**<ошибка>:=SYSTEM(SYS\_RESTxxx,0);**

где

- SYS\_RESTxxx может быть:
  - \* SYS\_RESTBOO для восстановления всех булевских переменных.
  - \* SYS\_RESTANA для восстановления всех аналоговых переменных.
  - \* SYS\_RESTTMR для восстановления всех таймерных переменных.
  - \* SYS\_RESTALL для восстановления всех булевских, аналоговых и таймерных переменных.
- <ошибка> получает значение ненулевого кода ошибки если операция не выполнялась. (SYS\_INITxxx не отработала).

Вот сводка команд функции SYSTEM для управления резервированием переменных

Команда		Значение
Ключевое слово	Значение	
SYS_INITBOO	16#20	Инициал. Резервирование булевских
SYS_SAVBOO	16#21	Сохранить булевские
SYS_RESTBOO	16#22	Восстановить булевские
SYS_INITANA	16#24	Инициал. Резервирование аналогов
SYS_SAVANA	16#25	Сохранить аналогии
SYS_RESTANA	16#26	Восстановить аналогии
SYS_INITTMR	16#28	Инициал. Резервирование таймеров

SYS_SAVTMR	16#29	Сохранить таймеры
SYS_RESTTMR	16#2A	Восстановить таймеры
SYS_INITALL	16#2C	Инициал. Резервирование всех типов
SYS_SAVALL	16#2D	Сохранить все типы
SYS_RESTALL	16#2E	Восстановить все типы

Команда (ключевое слово)	Аргумент	Возвр. Значение
SYS_INITxxx	адрес памяти	след. свободный адрес
SYS_SAVxxx	0	0 если успех
SYS_RESTxxx	0	0 если успех

### ▬ Пользовательская реализация

Наконец, используя функции “С” или функциональные блоки Вы можете построить процедуры доступа к памяти, питаемой от батареи, чтобы в любой момент сохранять и восстанавливать переменные в приложении.

#### Примеры:

- 1) Процедура, предназначенная для приложения:

backup, restore\_temp, restore\_date, restore\_cnt - “С” процедуры пользователя.

**backup**(temperature,date,cnt);      сохранение трех критических переменных

temperature=**restore\_temp**();      восстановление температуры

data=**restore\_date**();      восстановление даты

cnt=**restore\_cnt**();      восстановление счетчика

- 2) Процедура общего назначения:

backup\_init, backup\_link, restore - “С” процедуры пользователя.

save\_id=**backup\_init**(address,size);      выделить область в энергонезависимой памяти

**backup**(save\_id,cpt1,3);      сохранить cpt1 как третий элемент

rest\_id=**backup\_link**(address,size);      прилинковать выделенную память

cpt1=**restore**(rest\_id,3);      восстановить сохраненное значение

### С.9.3 Сохранение состояния программы

Существует возможность сохранения любого состояния любой прикладной программы, однако представляется рискованным восстанавливать программу в том состоянии, которое было зафиксировано в момент последнего резервирования, по крайней мере, по трем причинам:

- Некоторые процессы требуют определенных предварительных действий перед рестартом
- Очень нудно возиться со всеми многочисленными статусными характеристиками состояния процесса

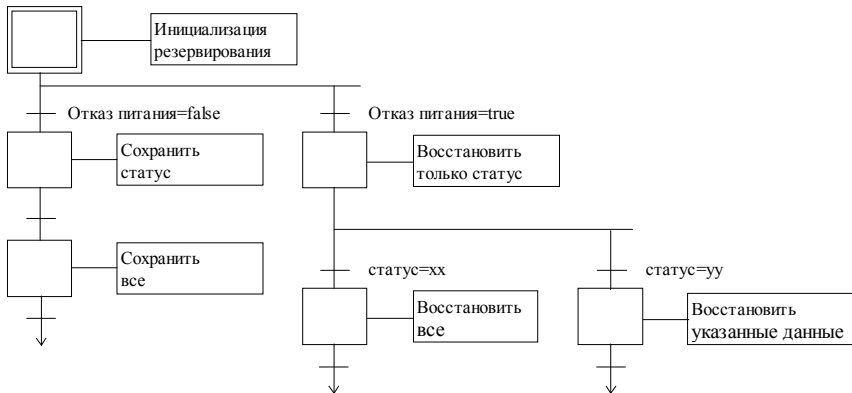
– Некоторые внешние ресурсы, такие как периферийные устройства или программы С не могут быть автоматически перезапущены

Представляется, что лучшее решение - это сохранить аналоговые или булевские переменные, описывающие состояние процесса, в те моменты, когда, по мнению программиста, есть возможность использовать их при рестарте. После этого было бы возможно с помощью возможно неполного, но достаточно разумного 'снимка' состояния процесса запустить, убить или заморозить программы SFC и инициализировать переменные, чтобы привести приложение в адекватное состояние. В ISaGRAF не предусмотрено никакой автоматической процедуры старта.

Затем можно из незавершенного, но интеллектуального 'образа' процесса запускать, убивать или замораживать программы SFC и инициализировать переменные, чтобы привести приложение в адекватное состояние. Но процедура автоматического запуска не может быть обеспечена ISaGRAF.

Представляется, что лучшее решение - это сохранить аналоговые или булевские переменные, описывающие состояние процесса, в те моменты, когда, по мнению программиста, есть возможность использовать их при рестарте. После этого было бы возможно с помощью возможно неполного, но достаточно разумного 'снимка' состояния процесса запустить, убить или заморозить программы SFC и инициализировать переменные, чтобы привести приложение в адекватное состояние. В ISaGRAF не предусмотрено никакой автоматической процедуры старта.

#### Пример:



## C.10 Приложение: Сообщения об ошибках и их описание

### Список ошибок:

Код	Сообщение	Тип
1	Не могу выделить память под базу данных для исполнения	система
2	Неправильная база данных приложения (Motorola/Intel)	приложение
3	Не могу выделить почтовый ящик для связи	система
4	Не могу привязать базу данных ядра	система
5	Таймаут по запросу к ядру	система
6	Таймаут по ответу от ядра	система
7	Не могу инициализировать связь	система
8	не могу выделить память под сохраняемые переменные	приложение
9	приложение остановлено	приложение
10	слишком много одновременных действий N или P	приложение
11	слишком много одновременных действий setting	приложение
12	слишком много одновременных действий resetting	приложение
13	неизвестная инструкция TIC	приложение
16	не могу выполнить запрос чтения данных	система
17	не могу выполнить запрос записи данных	система
18	не могу выполнить запрос сессии отладчика	система
19	не могу выполнить запрос modbus	система
20	не могу выполнить запрос приложения отладчика	система
21	не могу ответить отладчику	система
23	неизвестный код запроса	система
24	коммуникационная ошибка ethernet	система
25	коммуникационная ошибка синхронизации	система
28	не могу выделить память для приложения	система
29	не могу выделить память для обновления приложения	система
30	неизвестный код OEM	приложение
31	не могу инициализировать плату булевских входов	приложение
32	не могу инициализировать плату аналоговых входов	приложение
33	не могу инициализировать плату строковых входов	приложение
34	не могу инициализировать плату булевских выходов	приложение
35	не могу инициализировать плату аналоговых выходов	приложение
36	не могу инициализировать плату строковых выходов	приложение
37	не могу прочитать булевский вход	приложение
38	не могу прочитать аналоговый вход	приложение
39	не могу прочитать строковый вход	приложение
40	не могу вывести переменную булевский выход	приложение
41	не могу вывести переменную аналоговый выход	приложение
42	не могу вывести переменную строковый выход	приложение
43	не могу обработать булевскую переменную	приложение
44	не могу обработать аналоговую переменную	приложение

45	не могу обработать строковую переменную	приложение
46	не могу открыть плату	Приложение
47	не могу закрыть плату	Приложение
50	не могу переписать переменную булевский выход	Программа
51	не могу переписать переменную аналоговый выход	Программа
52	не могу переписать переменную строковый выход	Программа
61	неизвестный системный код запроса	Программа
62	переполнение периода выборки	Программа
63	функция пользователя не реализована	Приложение
64	деление целого на 0	Программа
65	функция преобразования не реализована	Приложение
66	функц. блок не реализован	Приложение
67	стандартная функция не реализована	Приложение
68	деление вещественного на 0	Программа
69	неправильные рабочие параметры	Приложение
72	параметры приложения нельзя модифицировать	Приложение
73	не могу обновить: отличается набор булевских переменных	Приложение
74	не могу обновить: отличается набор аналоговых переменных	Приложение
75	не могу обновить: отличается набор переменных типа timer	Приложение
76	не могу обновить: отличается набор переменных типа message	Приложение
77	не могу обновить: не могу найти новое приложение	Приложение
> 100	особый OEM код ошибки, за более детальной информацией обращайтесь к вашему поставщику.	

3 типа ошибок соответствуют различным источникам неисправностей:

**-Система:**

Такие проблемы, вероятно, обусловлены программным или аппаратным обеспечением целевой задачи, а не настройкой приложения или выполнением программы.

Попробуйте перезапустить вашу целевую задачу, или запустить другие приложения.

О таких ошибках следует докладывать вашей поддержке ISaGRAF.

**-Приложение:**

Такие проблемы обусловлены параметрами приложения, размером или содержанием.

Такие ошибки должны исчезать, когда загружается заранее отлаженное приложение.

Если проблема не исчезает, то она является системной ошибкой, как описано выше.

**-Программы:**

Такие проблемы обусловлены определенной последовательностью программы.

Такого сорта ошибки должны исчезать, когда приложение запускается в пошаговом режиме, или, когда программа установлена.

**Описание сообщений об ошибках:**

<b>1. не могу выделить память под базу данных для исполнения</b>	<b>система</b>
--	----------------

Не хватает оперативной памяти.

<b>2. неправильная база данных приложения (Motorola/Intel)</b>	<b>приложение</b>
--	-------------------

Неправильный формат сохраненного или загруженного файла приложения. Такая ошибка возникает если приложение было сгенерировано для Intel, а загружено в Motorola или наоборот или если файл был изменен.

<b>3. не могу выделить почтовый ящик для связи</b>	<b>система</b>
--	----------------

Это сообщение об ошибке возникает тогда, когда задача связи не может захватить память для объекта 3 для связи между задачами.

<b>4. не могу привязать базу данных ядра</b>	<b>система</b>
--	----------------

Это сообщение об ошибке выдается задачей связи если она не может найти целевой задачи с номером подчиненного определенным в командной строке.

<b>5. таймаут по запросу к ядру</b>	<b>система</b>
-------------------------------------	----------------

Задача связи не может послать запрос целевой задаче. Целевая задача отсутствует либо занята.

<b>6. таймаут по ответу от ядра</b>	<b>система</b>
-------------------------------------	----------------

Задача связи не может получить ответ от целевой задачи. Целевая задача отсутствует либо занята.

<b>7. не могу инициализировать связь</b>	<b>система</b>
--	----------------

Это сообщение выдается, когда уровень связи не может инициализировать физическую связь. Это предупреждение выдается также, когда не указан путь связи. Это сообщение не означает, что программа работает неправильно, но она не может связаться с подсистемой разработки.

<b>8. не могу выделить память под сохраняемые переменные</b>	<b>приложение</b>
--	-------------------

ISaGRAF не может работать с сохраняемыми переменными. Возможны две причины этого сообщения:

- строка, переданная как параметр целевой задаче синтаксически неверна
- размер памяти определенного для каждого блока недостаточен

Вы должны проверить синтаксис параметра сохраняемых переменных или попытаться уменьшить количество сохраняемых переменных.

<b>9. приложение остановлено</b>	<b>приложение</b>
----------------------------------	-------------------



Это сообщение выдается, если приложение было остановлено отладчиком.

<b>10. слишком много одновременных действий N или P</b>	<b>приложение</b>
---	-------------------

Это сообщение возникает, если целевая задача должна выполнять слишком много не запоминаемых, импульсных действий или циклических блоков одновременно. Максимальное количество одновременных действий 2 + 4 на программу SFC.

<b>11. слишком много одновременных действий setting</b>	<b>приложение</b>
---	-------------------

Это сообщение возникает, если один из циклов целевой задачи должен выполнить слишком много действий установки (выполняемых, когда шаг становится активным).

<b>12. слишком много одновременных действий resetting</b>	<b>приложение</b>
---	-------------------

Это сообщение возникает, если один из циклов целевой задачи должен выполнить слишком много действий сброса (выполняемых, когда шаг становится неактивным).

<b>13. неизвестная инструкция TIC</b>	<b>приложение</b>
---------------------------------------	-------------------

Целевая задача определила ошибку в коде приложения. Возможны два объяснения:  
- Внешняя программа пишет в код приложения. Попробуйте локализовать ошибку в режиме СС и убедитесь, что интерфейс В/В не имеет неверных параметров.  
- ваша целевая задача имеет сокращенный набор инструкций, и ваше приложение использует не существующие инструкции или типы переменных.

<b>16. не могу выполнить запрос чтения данных</b>	<b>система</b>
---	----------------

Определена ошибка связи при ответе на запрос ISaGRAF Modbus с кодом 18 (чтение файла). Проверьте связь и конфигурацию системы со стороны целевой задачи и подсистемы разработки.

<b>17. не могу выполнить запрос записи данных</b>	<b>система</b>
---	----------------

Определена ошибка связи при ответе на запрос ISaGRAF Modbus с кодом 17 (запись в файл). Проверьте связь и конфигурацию системы со стороны целевой задачи и подсистемы разработки.

<b>18. не могу выполнить запрос сессии отладчика</b>	<b>система</b>
--	----------------

Определена ошибка связи при ответе на запрос отладчика. Проверьте связь и конфигурацию системы со стороны целевой задачи и подсистемы разработки.

<b>19. не могу выполнить запрос modbus</b>	<b>система</b>
--	----------------

Определена ошибка связи при ответе на запрос Modbus. Проверьте связь и конфигурацию системы со стороны целевой задачи и подсистемы разработки.

<b>20. не могу выполнить запрос приложения отладчика</b>	<b>система</b>
--	----------------

Определена ошибка связи при ответе на запрос отладчика. Проверьте связь и конфигурацию системы со стороны целевой задачи и подсистемы разработки.

<b>21. не могу ответить отладчику</b>	<b>система</b>
---------------------------------------	----------------

Определена ошибка связи при ответе на запрос отладчика. Проверьте связь и конфигурацию системы со стороны целевой задачи и подсистемы разработки.

<b>23. неизвестный код запроса</b>	<b>система</b>
------------------------------------	----------------

Неверный запрос отладчика

<b>24. коммуникационная ошибка ethernet</b>	<b>система</b>
---	----------------

Сообщение возникает, если был закрыт отладчик. В этом случае программа работает правильно. В других случаях это означает, что была обнаружена ошибка связи по ETHERNET. Проверьте связь и конфигурацию системы со стороны целевой задачи и подсистемы разработки.

Второе поле означает:

1. Ошибка при отправке или приеме
2. Ошибка при создании сокета
3. Ошибка при связи или прослушивании сокета
4. Ошибка при приеме нового клиента

<b>25. коммуникационная ошибка синхронизации</b>	<b>система</b>
--	----------------

Плохая синхронизация задачи связи и подсистемой разработки. Проверьте связь и конфигурацию системы со стороны целевой задачи и подсистемы разработки.

<b>28. не могу выделить память для приложения</b>	<b>система</b>
---	----------------

Не хватает оперативной памяти. Сравните размер памяти приложения и размер свободной оперативной памяти.

<b>29. не могу выделить память для обновления приложения</b>	<b>система</b>
--	----------------

Не хватает оперативной памяти. Сравните размер памяти приложения и размер свободной оперативной памяти.

<b>30. неизвестный код OEM</b>	<b>приложение</b>
--------------------------------	-------------------

Приложение использует плату ключ, который не распознается целевой задачей. Проверьте соединение В/В в системе разработки и используйте атрибут 'VIRTUAL' чтобы локализовать неправильную плату. Ваша библиотека системы разработки может не соответствовать версии вашей целевой задачи.

<b>31. не могу инициализировать плату булевских входов</b>	<b>приложение</b>
--	-------------------

Неправильно инициализирована плата цифрового ввода. Проверьте связь ввода/вывода в подсистеме разработки и параметры Вашей платы цифрового ввода.

<b>32. не могу инициализировать плату аналоговых входов</b>	<b>приложение</b>
---	-------------------

Неправильно инициализирована плата аналогового ввода. Проверьте связь ввода/вывода в подсистеме разработки и параметры Вашей платы аналогового ввода.

<b>33. не могу инициализировать плату строковых входов</b>	<b>приложение</b>
--	-------------------

Неправильно инициализирована плата ввода сообщений. Проверьте связь ввода/вывода в подсистеме разработки и параметры Вашей платы ввода сообщений.

<b>34. не могу инициализировать плату булевских выходов</b>	<b>приложение</b>
---	-------------------

Неправильно инициализирована плата цифрового вывода. Проверьте связь ввода/вывода в подсистеме разработки и параметры Вашей платы цифрового вывода.

<b>35. не могу инициализировать плату аналоговых выходов</b>	<b>приложение</b>
--	-------------------

Неправильно инициализирована плата аналогового вывода. Проверьте связь ввода/вывода в подсистеме разработки и параметры Вашей платы аналогового вывода.

<b>36. не могу инициализировать плату строковых выходов</b>	<b>приложение</b>
---	-------------------

Неправильно инициализирована плата вывода сообщений. Проверьте связь ввода/вывода в подсистеме разработки и параметры Вашей платы вывода сообщений.

<b>37. не могу прочитать булевский вход</b>	<b>приложение</b>
---	-------------------

Ошибка при чтении платы цифрового ввода. Проверьте связь ввода/вывода в подсистеме разработки и параметры Вашей платы цифрового ввода.

<b>38. не могу прочитать аналоговый вход</b>	<b>приложение</b>
--	-------------------

Ошибка при чтении платы аналогового ввода. Проверьте связь ввода/вывода в подсистеме разработки и параметры Вашей платы аналогового ввода.

<b>39. не могу прочитать строковый вход</b>	<b>приложение</b>
---	-------------------

Ошибка при чтении платы ввода сообщений. Проверьте связь ввода/вывода в подсистеме разработки и параметры Вашей платы ввода сообщений.

<b>40. не могу вывести переменную булевский выход</b>	<b>приложение</b>
---	-------------------

Ошибка при записи платы цифрового вывода. Проверьте связь ввода/вывода в подсистеме разработки и параметры Вашей платы цифрового вывода.

**41. не могу вывести переменную аналоговый выход****приложение**

Ошибка при записи аналогового вывода. Проверьте связь ввода/вывода в подсистеме разработки и параметры Вашей платы аналогового вывода.

**42. не могу вывести переменную строковый выход****приложение**

Ошибка при записи платы вывода сообщений. Проверьте связь ввода/вывода в подсистеме разработки и параметры Вашей платы вывода сообщений.

**43. не могу обработать булевскую переменную****приложение**

Ошибка при исполнении вызова OPERATE для булевой переменной. Проверьте параметры вашего вызова OPERATE.

**44. не могу обработать аналоговую переменную****приложение**

Ошибка при исполнении вызова OPERATE для аналоговой переменной. Проверьте параметры вашего вызова OPERATE.

**45. не могу обработать строковую переменную****приложение**

Ошибка при исполнении вызова OPERATE для переменной сообщения. Проверьте параметры вашего вызова OPERATE.

**46. не могу открыть плату****приложение**

Приложение использует ссылку на плату неизвестную целевой задаче. Проверьте связь ввода/вывода в подсистеме разработки. Возможно, библиотека подсистемы разработки не соответствует вашей версии целевой задаче.

**47. не могу закрыть плату****приложение**

Приложение использует ссылку на плату неизвестную целевой задаче. Проверьте связь ввода/вывода в подсистеме разработки.

**50. не могу переписать переменную булевский выход****программа**

Две SFC последовательности пишут одну и ту же переменную цифрового вывода в одном цикле. Этого нужно избегать. В случае такого конфликта приоритет имеет программа занимающая более высокое место в иерархии. Если две программы размещены на одном и том же уровне результат непредсказуем.

**51. не могу переписать переменную аналоговый выход****программа**

Две SFC последовательности пишут одну и ту же переменную аналогового вывода в одном цикле. Этого нужно избегать. В случае такого конфликта приоритет имеет программа

занимающая более высокое место в иерархии. Если две программы размещены на одном и том же уровне результат непредсказуем.

<b>52. не могу переписать переменную строковый выход</b>
--

<i>программа</i>
------------------

Две SFC последовательности пишут одну и ту же переменную вывода сообщений в одном цикле. Этого нужно избегать. В случае такого конфликта приоритет имеет программа занимающая более высокое место в иерархии. Если две программы размещены на одном и том же уровне результат непредсказуем.

<b>61. неизвестный системный код запроса</b>
--

<i>программа</i>
------------------

Программа использует неправильный системный вызов.

<b>62. переполнение периода выборки</b>
---

<i>программа</i>
------------------

Цикл целевой задачи дольше, чем он определен в подсистеме разработки. В многозадачной системе это означает что для выполнения цикла не хватает времени даже если текущая длительность цикла меньше чем определенный период. В однозадачной системе это всегда означает, что на один цикл целевой задачи приходится слишком много операций.

Существует несколько способов устранения этого сообщения:

- уменьшить количество операций в том месте, где было выдано сообщение
- уменьшить количество переходов, оптимизировать сложную обработку и т.д.
- уменьшить загрузку процессора другими задачами для того, чтобы дать больше времени ISaGRAF
- уменьшить нагрузку на связь для того, чтобы дать больше времени ISaGRAF
- использовать динамическую модификацию длительности циклов, чтобы адаптировать длительность циклов к различным этапам процесса
- установить нулевую длительность цикла и дать возможность ISaGraf так быстро как она может без контроля превышения времени цикла.

<b>63. функция пользователя не реализована</b>
--

<i>приложение</i>
-------------------

Программа использует неизвестную целевой задаче C функцию. Ваша библиотека подсистемы разработки может не соответствовать версии целевой задачи.

<b>64. деление целого на 0</b>
--------------------------------

<i>программа</i>
------------------

Программа пытается разделить целое число на 0. Когда такое случается ISaGRAF использует в качестве результата максимальное число. Если операнд отрицательный то число инвертируется.

Когда это случается, ISaGRAF использует в качестве результата максимальное аналоговое значение.

Если операнд отрицательный, результат инвертируется.

<b>65. функция преобразования не реализована</b>
--

<i>приложение</i>
-------------------

Программа использует С функцию преобразования неизвестную целевой задаче. Ваша библиотека подсистемы разработки может не соответствовать версии целевой задачи. Когда такое случается ISaGRAF не преобразует значение.

<b>66. функц. блок не реализован</b>
--------------------------------------

<i>приложение</i>
-------------------

Программа использует С функциональный блок неизвестный целевой задаче. Ваша библиотека системы разработки не соответствует версии вашей целевой задачи.

<b>67. стандартная функция не реализована</b>
---

<i>приложение</i>
-------------------

Программа использует функциональный блок неизвестный целевой задаче.

<b>68. деление вещественного на 0</b>
---------------------------------------

<i>программа</i>
------------------

Программа пытается разделить действительное число на 0. Когда такое случается ISaGRAF использует в качестве результата максимальное число. Если операнд отрицательный то число инвертируется.

Когда это случается, ISaGRAF использует в качестве результата максимальное аналоговое значение.

Если операнд отрицательный, результат инвертируется.

<b>69. неправильные рабочие параметры</b>
---

<i>приложение</i>
-------------------

Ваше приложение использует вызов OPERATE с неправильными параметрами. Обычно это отфильтровывается компилятором. Это может быть таймер либо внутренняя переменная.

<b>72. параметры приложения нельзя модифицировать</b>
---

<i>приложение</i>
-------------------

При попытке модифицировать приложение, модифицированное приложение не может начать работу так как символы отличаются. Возможно одна или несколько переменных, или переходов, или функциональных блоков были добавлены, удалены или модифицированы по сравнению с текущим приложением.

<b>73. не могу обновить: отличается набор булевских переменных</b>
--

<i>приложение</i>
-------------------

Модифицированное приложение не может начать работу, потому что некоторые булевы переменные были добавлены или удалены, по сравнению с текущим приложением.

<b>74. не могу обновить: отличается набор аналоговых переменных</b>
---

<i>приложение</i>
-------------------

Модифицированное приложение не может начать работу, потому что некоторые аналоговые переменные были добавлены или удалены, по сравнению с текущим приложением.

<b>75. не могу обновить: отличается набор переменных типа timer</b>
---

<i>приложение</i>
-------------------

Модифицированное приложение не может начать работу, потому что некоторые переменные-таймеры были добавлены или удалены, по сравнению с текущим приложением.

<b>76. не могу обновить: отличается набор переменных типа message</b>	<b>приложение</b>
---	-------------------

Модифицированное приложение не может начать работу, потому что некоторые переменные-сообщения были добавлены или удалены, по сравнению с текущим приложением.

<b>77. не могу обновить: не могу найти новое приложение</b>	<b>приложение</b>
---	-------------------

Модифицированное приложение не найдено в памяти, что-то неверное случилось при загрузке.

## D. Глоссарий

Action	Действие	Список операторов или присваиваний выполненных за один шаг программы SFC в активном состоянии.
Action (FC)	Действие (FC)	Символ Поточковой Диаграммы. Действие представляет собой список инструкций, которые должны быть выполнены , кода динамический поток встречает символ действия.
Activity of a step	Активность шага	Атрибут шага, который определяется выражением SFC. Действия, связанные с шагом, выполняются в соответствии с их активностью.
Analog Attribute	Аналоговый Атрибут	Тип переменной. Это длинное целое или вещественное число. Класс переменных. Возможные атрибуты переменных - внутренние, ввода или вывода.
Begin section	Начальная секция	Группа последовательных программ, выполняемых в начале каждого цикла запуска.
Beginning step	Начальный шаг	Первый шаг в теле макроса. Начальный шаг не связан с какими-либо предыдущими действиями.
Boolean	Булевский тип	Тип переменных. Переменные данного типа могут принимать только истинное или ложное значение
Boolean action	Булевское действие	Действие SFC: булевской переменной присваивается значение активности сигнала шага
Breakpoint	Точка останова	Место в программе, помеченное пользователем во время отладки. Точка останова - шаг SFC или переход. Система останавливается, когда SFC достигает Точки останова.
C function	«C»-функция	Функция, написанная на «C», вызванная из программы ISaGRAF (написанной на другом языке) синхронно. «C»-функции поставляются CJ International или пишутся пользователем
C language	Язык «C»	Язык программирования высокого уровня, предназначенный для описания действий компьютера, таких как «C»-функции и функции преобразования
C source code	«C»-код	Текстовый файл, содержащий программу на языке «C»
C source header	Заголовочный файл «C»	Текстовый файл, содержащий прототипы функций и определения типов, необходимых для программы на языке «C»
Cell	Ячейка	Элементарная область в графической матрице для графических языков, таких как SFC, FBD или LD.
Child SFC program	Дочерняя SFC программа	SFC- программа, управляемая другой SFC- программой, называемой родительской
Clearing a transition	Освобождение перехода	Операция во время выполнения программы: все маркеры, созданные ранее, удаляются. Выражение создается на каждом следующем шаге.



Coil	Виток	Графическая составляющая LD-программы, используемая для отображения присваивания переменных вывода.
Comment	Комментарий	Текст, включенный в программу, но не имеющий влияния на её выполнение
Comment (SFC)	Комментарий (SFC)	Текст, присоединённый к SFC- шагу или переходу, но не имеющий влияния на её выполнение
Common	Общий	Диапазон объявления слов. Такие объекты могут быть использованы любой программой из любого проекта.
Condition (for a transition)	Условия ( для перехода )	Булево выражение присоединённое к SFC- переходу. Переход не может быть совершён, если выражение ложно.
Connector (FC)	Соединитель (FC)	Графическая компонента FC, которая представляет связь, от точки потоковой диаграммы действию ил тесту FC. Графический символ прыжка – это маленький кружок с номером ссылки элемента назначения.
Constant expression	Константа	Буквенное выражение, используемое для описания постоянного значения. Константы связаны с одним типом.
Contact	Контакт	Графическая составляющая LD- программы. Она отображает статус переменной ввода.
Conversion	Преобразование	Фильтр, связанный с аналоговой переменной ввода или вывода. Преобразование автоматически производится всякий раз, когда значение переменной вводится или выводится.
Conversion function	Функция преобразования	«С»- функция, которая описывает преобразование. Такое преобразование может быть присоединено к любой аналоговой переменной ввода / вывода
Conversion table	Таблица преобразования	Набор точек, которые определяют линейное (посегментное) преобразование. Такое преобразование может быть присоединено к любой аналоговой переменной ввода / вывода
Cross references	Перекрёстная ссылка	Информация, собранная системой разработки ISaGRAF, про словарь переменных, а также про их использование в программе.
Current result (IL)	Текущий результат (в языке IL)	Результат инструкции в IL-программе. Текущий результат может быть изменён инструкцией или использоваться для создания переменной.
Cycle timing	Временной цикл	Длительность исполнения текущего цикла.0000
Cycle to cycle mode	Режим последовательного выполнения циклов	Выполняемый режим: в этом режиме циклы исполняются по очереди, в соответствии с инструкциями пользователя, данными отладчику.
Cyclic	Циклический	Атрибут программы, которая всегда выполняется.
Decision (FC)	Решение (FC)	(Также называется тест) Символ потоковой диаграммы присоединённый к булевскому выражению. Поток направляется по символам Да или Нет в зависимости от состояния выражения.
Defined word	Макроопределение	Уникальный идентификатор, который используется для подстановки в любое выражение текста программы.

Delayed operation (IL)	Операция задержки (в языке IL)	Операция в IL-программе, которая выполняется при обнаружении инструкции «(» в программе.
Diary	Дневник	Текстовый файл, содержащий все записи об изменениях, сделанных над одной программой. Каждая запись завершается датой изменения.
Dictionary	Словарь	Список внешних переменных, переменных ввода/вывода и макросов, используемых в программах проекта.
Edge	Фронт	Изменение значения булевой переменной. Передний фронт означает изменение с ложного значения на истинное, задний фронт - с истинного на ложное.
End section	Секция End	Группа циклических программ, исполняемая в конце каждого указанного цикла.
Ending step	Последний шаг	Последнее действие в теле макро шага SFC.
Expression	Выражение	Набор операторов и идентификаторов, которые отображают вычисление формулы.
Father SFC program	Родительская SFC-программа	SFC-программа, которая управляет другой SFC- программой, называемой дочерней.
FBD FC	FBD FC	Язык диаграмм функциональных блоков "Потоковая Диаграмма".
Flow Chart	Потоковая Диаграмма	Графический язык использующийся для описания потоков. Диаграмма состоит из действий, которые должны выполняться и решений, которые допускаю выбор из различных направлений потока. Язык Потоковых Диаграмм позволяет рисовать цтклы, которые должны выполняться на последовательных циклах.
Function block	Функциональный блок	Графический компонент языка FBD, который представляет стандартную элементарную функцию библиотеки ISaGRAF.
Functional Block Diagram	Диаграмма функционального блока.	Графический язык : равенства состоят из стандартных элементарных блоков библиотеки ISaGRAF и связаны вместе в диаграмму.
Global	Глобальный	Диапазон области видимости переменных и макросов. Глобальные Объекты могут быть использованы любой программой, входящей в проект.
Hierarchy	Иерархия	Архитектура проекта, разделённого на несколько программ. Иерархическое дерево показывает связи между родительскими и дочерними программами.
I/O board	Плата ввода/вывода	Аппаратный ресурс. Плата характеризуется типом и направлением ( ввод или вывод ). Параметры платы описаны в библиотеке ISaGRAF.
I/O channel	Канал ввода/вывода	Точка соединения платы ввода - вывода. Канал может получать одну переменную ввода - вывода.
I/O connection	Соединение ввода/	Определение связей между переменными программы и каналами платы, существующими в выбранной системе.

I/O variable	вывода Переменная ввода/ вывода	Переменная, соединённая с устройством ввода или вывода. Переменная ввода - вывода должна быть соединена с каналом платы ввода - вывода.
Identifier	Идентификатор	Уникальное слово, используемое для представления переменной или константы в программировании.
IL	Список Инструкций	Язык Списка Инструкций
Initial situation	Состояние инициализации	Список начальных действий SFC- программы, которые определяют состояние программы в момент её старта.
Initial step	Шаг инициализации	Специальная операция SFC-программы, которая выполняется в момент запуска.
Input	Ввод	Атрибут переменной. Такая переменная может быть связана с устройством ввода.
Instruction	Инструкция	Элементарная операция IL-программы, внесённая в текстовую строку.
Instruction List	Список Инструкций	Язык низкого уровня, состоящий из последовательного списка элементарных операций.
Integer	Целое	Класс аналоговых переменных, хранящихся в формате 32-битового целого со знаком.
Internal	Внутренняя	Атрибут переменной, не связанной с устройством ввода - вывода.
Jump to a step	Прыжок к шагу	Графический компонент SFC, который изображает связь перехода с выполнением операции. Графический символ прыжка - стрелка с номером действия, на которое будет осуществлён прыжок.
Keyword	Ключевое слово	Зарезервированное слово языка программирования.
Label (IL)	Метка (в IL)	Идентификатор в начале IL-инструкции, который может быть использован как операнд оператора JMP.
Ladder Diagram	Релейная диаграмма	Графический язык, объединяющий контакты и витки. Предназначен для проектирования булевских уравнений.
LD		Язык релейных диаграмм
Level 1 of the SFC	1- уровень в SFC	Главное описание SFC-программы. 1-уровень группирует схемы и присоединяет комментарии.
Level 2 of the SFC	2- уровень в SFC	Детальное описание SFC-программы. В этом описании действия внутри шагов и булевские условия связываются с переходами.
Library	Библиотека	Набор программных и аппаратных ресурсов, которые могут быть непосредственно вставлены в программу.
Local	Локальный	Диапазон области видимости переменных и макросов. Локальные объекты могут быть использованы только одной из программ проекта.
Locked I/O	Закрытый ввод/ вывод	Переменная ввода или вывода, логически разъединённая с сообщаемым устройством ввода или вывода при помощи команды «Закрыть», данной пользователем в отладчике.
Macro step	Макро шаг	Графический компонент SFC. Макро шаг - это уникальная группа шагов и переходов, представленная определённым символом в главной диаграмме и описанная отдельно.
Matrix	Матрица	Логическое разделение редактируемой области на

		прямоугольные ячейки во время выполнения программы на графическом языке.
Message	Сообщение	Тип переменной. Переменные этого типа содержат строку переменной длины.
Modbus	Сетевой протокол	Master-Slave протокол. Целевая система ISaGRAF может быть подчиненным по отношению к внешней системе (такой как управляющая система ) в полной архитектуре.
Modifier (IL)	Модификатор ( в IL )	Одиночный символ, помещённый в конце ключевого слова IL, который изменяет значение оператора.
Network address	Сетевой адрес	Формальный шестнадцатеричный адрес, который может быть определён для каждой переменной. Этот адрес используется протоколом Modbus при соединении системы с внешней системой.
Non-stored action	Не сохраняемое действие	SFC- действие : список операторов, выполняемых в каждом цикле целевой задачи, когда соответствующий шаг - в активном состоянии.
OEM key code (I/O board)	Код изготовителя оборудования	Шестнадцатеричный 16-битовый код, определённый для каждой платы ввода - вывода в библиотеке ISaGRAF. Он идентифицирует изготовителя платы.
OEM parameter (I/O board)		Идентификационный параметр платы, определенный её разработчиком. Это может быть как константа, так и переменная, определяемая пользователем во время соединения ввода-вывода.
Operand (IL)	Операнд	Переменная или константа, обрабатываемая элементарной инструкцией IL.
Operation (IL)	Операция	Основная инструкция языка IL. В основном связана с операндами в инструкциях.
Output	Вывод	Атрибут переменной. Такие переменные связаны с устройством вывода целевой машины.
Parameter (C function)	Параметр «С» функции	Значение, переданное «С»- функции. П. Характеризуется его типом.
Parameter (I/O board)	Параметр платы ввода/вывода	Константа или определённое пользователем значение - параметр стандартной платы ввода-вывода. Может быть задано программно во время соединения ввода-вывода.
Parent program	Родительская программа	Программа, написанная на любом языке, которая управляет ( или вызывает ) другую программу ( написанную не на SFC ), называемую её подпрограммой.
Power rail	Шина питания	Главные левые и правые вертикальные шины, расположенные с краю релейной диаграммы.
Program	Программа	Основной программный элемент проекта. Программа пишется на одном языке и помещается в иерархическое дерево проекта.
Project	Проект	Программная область, которая содержит всю информацию ( программы, переменные, ... ) для одного приложения ISaGRAF.
Pulse action	Импульсное действие	SFC- действие : список операторов, выполняемых только один раз при активизации соответствующего шага.

Range	Диапазон	Часть программы, в которой может использоваться объект. Предопределённые диапазоны ISaGRAF - общий, глобальный, локальный.
Real	Вещественный	Класс аналоговой переменной, которая хранится как число с плавающей точкой с одинарной точностью в 32-битовом формате.
Real board	Реальная плата	Плата ввода-вывода, физически соединённая с устройством ввода-вывода целевого компьютера.
Real time mode	Режим реального времени	Нормальный режим исполнения: целевой цикл переключается при помощи программного временного цикла.
Reference number (SFC)	Номер ссылки	Десятичное число ( 1...65535 ), которое идентифицирует SFC-шаг или SFC-переход.
Register (IL)	Регистр	Текущий результат последовательности действий IL.
Return value of a sub-program	Возвращаемое значение подпрограммы.	Значение, возвращаемое подпрограммой в конце выполнения. Оно используется в операциях вызывавшей подпрограммы.
Run time error	Ошибка при выполнении	Ошибка приложения, обнаруженная ISaGRAF во время выполнения.
Section	Секция	Группа программ, выполняемая по одинаковым динамическим правилам.
Separator	Разделитель	Специальный символ или группа символов, используемая для разделения идентификаторов в языке.
Sequential Function Chart	Последовательная функциональная диаграмма	Графический язык : процесс описывается как список шагов, связанных переходами. Действия связаны с шагами. Переходы определяются булевыми условиями.
Sequential section	Последовательный раздел	Группа программ проекта. Выполнение этих программ подчинено динамическим правилам языка SFC.
SFC		Язык Последовательных Функциональных Схем
ST		Язык Структурированного текста
Statement Step	Оператор Шаг	Основная полная операция ST. Основной графический компонент SFC. Шаг представляет устойчивую стадию процесса и изображается квадратом. Каждому шагу ставится в соответствие его номер. Активность шага используется для контроля выполнения соответствующего процесса.
String	Строка	Набор символов, хранящийся в переменной типа message.
Structured Text	Структурированный текст	Структурированный язык высокого уровня, сочетающий присваивания, конструкции высокого уровня ( такие как if/then/else ) и вызовы функций.
Sub-program	Подпрограмма	Программа, написанная на каком-либо языке и вызываемая другой программой, называемой её родительской.

Target	Цель	Целевая машина ISaGRAF, которая поддерживает базовое ПО ISaGRAF.
Target cycle	Целевой цикл	Набор действий, выполняемый каждый раз, когда активизируется система ISaGRAF. Действия переключаются в соответствии с программным временным циклом.
Technical note	Технические примечания	Руководство пользователя по элементам библиотеки ISaGRAF ( «С»- функции, функциональные блоки, функции преобразования или платы ввода-вывода ).
Test (FC)	Тест (FC)	(Также называется решение)Символ потоковой диаграммы присоединенный к булевскому выражению. Поток направляется по символам Да или Нет в зависимости от состояния выражения.
Timer	Таймер	Тип переменной. Такие переменные содержат значение времени и могут быть автоматически обновлены системой во время выполнения программы.
Token (SFC)	Маркер	Графический маркер, используемый для изображения активного шага SFC- программы.
Toolbox	Комплект инструментов	Маленькое графическое окно, принадлежащее окну инструмента графического редактирования. Оно группирует главные клавиши для выбора графических компонентов.
Top level program	Программа верхнего уровня	Программа, размещённая на верху иерархического дерева. Вызывается системой.
Transition	Переход	Главный графический компонент SFC. Переход определяет отношение между различными SFC-шагами. Переход имеет идентификационный номер. К каждому переходу присоединяется булевское условие.
Type	Тип	Класс переменных одинакового формата. Основные типы - булевский, аналоговый, таймер и сообщение.
Validity of a transition Variable	Истинность перехода Переменная	Атрибут перехода. Переход проверяется, когда все предыдущие шаги активны. Уникальное представление элементарных данных, которое обрабатывается в программе или проекте.
Virtual board	Виртуальная плата	Плата ввода- вывода , которая физически не соединена с устройством ввода-вывода целевой машины.



## Е. Предметный указатель

% .....	A-103, B-209	FC .....	A-51, B-234, D-500
:= .....	<b>A-155</b>	FC подпрограмма .....	B-236
:= (СТ присвоение) .....	<b>B-265</b>	FC соединитель .....	A-54
=1 .....	B-288	FIND .....	B-356
1 уровень SFC .....	D-501	FM_READ .....	B-376
AnyTarget .....	A-117	From .....	A-118
appli.tst .....	C-386, C-420	<b>Goto</b> .....	<b>A-158</b>
appli.x8m .....	C-386, C-420	<b>If A-159</b>	
ARCREATE .....	B-364	IF .....	B-238
ARREAD .....	B-365	IL .....	A-136, D-501
ARWRITE .....	B-366	INSERT .....	B-357
ASCII .....	B-353	Integer .....	A-91
ASIN .....	B-337	ISA задача (OS9) .....	C-389
Begin .....	A-27	isa_main .....	C-401, C-404
BinaryFile .....	A-116	isa_register_slave .....	C-400
Boolean .....	A-91, D-498	ISAKER задача (OS9) .....	C-390
С заголовков .....	D-498	ISAKERET.O (VxWorks) .....	C-402
С код .....	A-113	ISAKERSE.O (VxWorks) .....	C-402
С функции .....	C-431, C-438	ISAMOD (VxWorks) .....	C-399
С функциональные блоки .....	C-431	ISANET задача (OS9) .....	C-391
С функциональный блок .....	A-169	ISATST задача (OS9) .....	C-390
С функция .....	A-169, D-498	ISAx0 .....	C-396
CHAR .....	B-354	ISAx1 .....	C-396
<b>Cycle</b> .....	<b>A-157</b>	ISAx1 (VxWorks) .....	C-409
DAY_TIME .....	B-363	ISAx2 .....	C-396
DDE .....	A-132	ISAx3 .....	C-396
DELETE .....	B-355	ISAx4 .....	C-396
EN .....	A-62	ISAx5 .....	C-396
End .....	A-27, <b>A-159</b>	ISAx6 .....	C-396
ENO .....	A-62	<b>Label</b> .....	<b>A-158</b>
Ethernet .....	A-37	LD .....	A-48, A-58, A-60, D-501
F_CLOSE .....	B-368	LEFT .....	B-358
F_EOF .....	B-369	LIMIT .....	B-346
F_ROPEN .....	B-367	MAX .....	B-345
F_WOPEN .....	B-367	Message .....	A-91
FA_READ .....	B-371	MID .....	B-359
FA_WRITE .....	B-373	MIN .....	B-345
FALSE .....	A-91	MLEN .....	B-360
FBD .....	A-69, C-439, D-500	MOD .....	B-347



Modbus .....	D-502	<b>Wait</b> .....	<b>A-157</b>
MODBUS .....	A-94	WHILE .....	B-238
MUX4 .....	B-348	XOR .....	B-288
MUX8 .....	B-349	Активизировать .....	A-124
N модификатор .....	A-47	Активность шага .....	B-270, D-498
NOT .....	A-71, A-73	Активность шага .....	B-214
ODD .....	B-350	Аналог .....	C-433, D-498
OEM ключ .....	A-166	Аналоги .....	C-432
OEM параметр .....	A-166	Аргумент .....	A-168
OEM параметр (плата В/В) .....	D-502	Арксинус .....	B-337
OR .....	A-70	Архив .....	A-25, A-164, A-171, A-180
P модификатор .....	A-46	Атрибут .....	D-498
P0 модификатор .....	A-47	Библиотека ...A-25, A-32, A-33, A-101,	
P1 модификатор .....	A-47	A-102, A-121, A-149, <b>A-161</b> , A-171,	
<b>Print</b> .....	<b>A-156</b>	C-431, D-502	
<b>PrintTime</b> .....	<b>A-156</b>	Бинарный селектор .....	B-352
Quick LD .....	A-48, A-58, A-60	Битмап .....	A-138
RAND .....	B-351	Битовое поле .....	A-140
Real .....	A-91	Блокировать .....	A-126
REPEAT .....	B-238	Блокировка .....	A-182
REPLACE .....	B-361	Булевское действие .....	D-498
Return .....	A-61, A-71	Булевское действие .....	A-47
RIGHT .....	B-362	В/В A-34, A-100, A-101, A-122, A-126,	
Run-time .....	A-33	A-148, A-164, A-165, A-182	
SEL .....	B-352	В/В специфический .....	B-237
Sequential .....	A-27	Ввод .....	A-166
SFC A-38, A-112, A-128, A-176, C-439,		Ввод .....	A-100, A-122, A-148, A-150
D-504		Верхний уровень .....	A-27
SFC галерея .....	A-49	Виртуальная плата .....	A-182, D-505
SHL .....	B-343	Виртуальная плата .....	A-101
SHR .....	B-344	Виток .....	A-60, A-70, D-499
SIN .....	B-340	Вклеить LD .....	A-66
SlavesLink .....	C-406	Внутренний .....	D-501
ST .....	A-48, A-136, C-448, D-504	Возвращаемое значение .....	D-503
SYSTEM .....	B-308	Восстановить ...A-25, A-164, A-171, A-	
Target .....	A-117	173, A-180	
TextFile .....	A-117	Временной цикл .....	D-499
Timer .....	A-91	Временной цикл .....	A-33, A-126
To .....	A-118	Вставить FBD .....	A-74, A-75
TP .....	B-321	Вставить FC .....	A-56
TRUE .....	A-91	Вставить SFC .....	A-44
tst_main_ex .....	C-404	Вставить виток .....	A-64
ULongData .....	A-115	Вставить контакт .....	A-64
VarList .....	A-116	Вставить переменную .....	A-46, A-90

Вставить подстроку .....	B-357	Диапазон .....	D-503
Вставить слот .....	A-101	Директория .....	A-187
Вставить ступень .....	A-66	Длина сообщения .....	B-360
Вставить текст .....	A-78	Длина строки .....	B-360
Вставить файл .....	A-79	Длительность активности .....	B-214, B-270
Вставить элемент FBD .....	A-72	Дневник .....	A-30, D-500
Вставить элемент FC .....	A-53	Документ .....	A-24, A-35, A-174
Встроенный исходный код .....	A-145	Документ проекта .....	A-24
Вход .....	B-203, D-501	Документ проекта .....	A-35, A-174
Выбрать Прожектор .....	A-140	Дочерняя .....	B-199
Выбрать элемент FBD .....	A-72	Дочерняя SFC .....	B-199
Выбрать элемент FC .....	A-53	Дочерняя программа .....	A-28
Вывод .....	A-100, A-122, A-148, A-166	Дочерняя программа SFC .....	A-28, D-499
Выгрузить .....	A-144	Заголовки C .....	C-434, C-441
Выгрузить (подготовить) .....	A-145	Заголовок C .....	C-465
Выполнить один цикл .....	A-126	Задержанная операция (IL) .....	D-500
Выражение .....	D-500	Закрытый В/В .....	D-502
Вырезать FBD .....	A-74	Закрыть файл .....	B-368
Вырезать FC .....	A-56	Заменить A-45, A-56, A-67, A-74, A-78	
Вырезать LD .....	A-66	Заменить подстроку .....	B-361
Вырезать SFC .....	A-44	Запись в файл .....	B-373
Вырезать переменную .....	A-90	Запись массива .....	B-366
Вырезать текст .....	A-78	Защита .....	A-24, A-105, A-162, A-178, A-180
Выход .....	B-203, D-502	Идентификатор .....	D-501
Галерея .....	A-49	Иерархия .....	A-27, A-31, B-198, D-501
Генерация кода .....	A-33, A-110	Извлечение подстроки (в середине) .....	B-359
Глобальный .....	D-500	Извлечение подстроки (слева) .....	B-358
Графика .....	A-138, A-143	Извлечение подстроки (справа) .....	B-362
Группа .....	A-24, A-140	Измененный стиль .....	A-76
Группа проектов .....	A-24	Изменить .....	A-125
Дамп .....	A-135	Изменить переменную .....	A-90
Двигать FBD .....	A-72	Изменить размер FC .....	A-55
Двигать FC .....	A-55	Изменить размер Прожектора .....	A-140
Двигать SFC .....	A-44	Иконка .....	A-140
Двигать плату .....	A-101	Импорт .....	A-95
Двигать Прожектор .....	A-140	Импортировать функциональный блок .....	A-32
Двинуть программу .....	A-31	Импортировать функцию .....	A-32
Действие .....	A-51, B-237, D-498	Импульс .....	A-46
Действительная плата .....	A-101	Импульсное действие .....	D-503
Действительный .....	D-503	Импульсное таймирование .....	B-321
Действия .....	B-235	Инвертированная связь .....	A-71, A-73
Дескриптор .....	A-22, A-34		
Дескриптор проекта .....	A-23		
Диагностика .....	A-147		

Инструкция .....	D-501	Локальная .....	A-168
Интерфейс.....	A-31, A-168	Локальный .....	D-502
Истинность перехода .....	D-504	Макроопределение .....	D-500
История.....	A-23, A-35	Макрос .....	A-87, A-91
Исходник C ....	C-435, C-442, C-452, C-465	Макрошаг .....	D-502
Исходный код C .....	D-498	Макрошаг .....	A-41, A-43
Канал.....	A-102, A-104, A-166	Максимум .....	B-345
Канал В/В .....	D-501	Маркер (SFC) .....	D-504
Ключевое слово .....	D-501	Масштаб .....	A-59, A-67, A-76
ключевой код OEM.....	D-502	Матрица .....	D-502
Колонка.....	A-138	Мдификатор (IL).....	D-502
Комментарий .....	B-238, D-499	Менеджер библиотек.....	A-161
комментарий FBD.....	A-72	Менеджер библиотек.....	C-433, C-447
Комментарий FC.....	A-54	Менеджер программ.....	A-27
Комментарий программы .....	A-30	Меню инструменты .....	A-35
Комментарий ступени.....	A-63, A-67	Метафайл.....	A-138
Комметарий (SFC).....	D-499	Метка .....	A-71, B-242, B-257
Коммуникация A-37, A-126, A-144, A-186		Метка (IL).....	D-501
Компилировать A-33, A-110, A-163, A-167		Метка ступени .....	A-64
Компилятор C .....	C-431, C-465	Минимум .....	B-345
Конец .....	B-234	Модификация на ходу .....	A-125
Конечный шаг.....	A-43	Модификация на ходу .....	A-128
Константа.....	D-499	Модуль.....	B-347
Контакт .....	A-60, A-70, D-499	Мультиплексор с 4 входами .....	B-348
Контакт с отрицательным фронтом B-250		Мультиплексор с 8 входами .....	B-349
Контакт с положительным фронтом B-250		На линии .....	A-36
конфигурация В/В.....	A-164	На ходу .....	A-123
Конфигурация В/В.....	A-22	Набор инструментов.....	D-504
Копировать FBD .....	A-74	Найти A-45, A-56, A-67, A-74, A-78, A-85	
Копировать FC.....	A-56	Найти подстроку.....	B-356
Копировать LD.....	A-66	Начало .....	B-234
Копировать SFC.....	A-44	Начальная ситуация.....	D-501
Копировать переменную .....	A-90	Начальный шаг .....	D-498, D-501
Копировать программу.....	A-32	Начальный шаг .....	A-41, A-43
Копировать текст .....	A-78	Не сохраненное.....	A-47
Копия .....	A-164, A-171, A-172, A-180	Не сохраненное действие .....	D-502
Коснись .....	A-33	Непосредственно представленные переменные .....	A-103, B-209
Кривая.....	A-138, A-139, A-143	Новая переменная.....	A-89
Крснись .....	A-111	Новая программа .....	A-29
		Новая ступень .....	A-63
		Новая функция.....	A-29
		Новый проект .....	A-22

Новый функциональный блок ..... A-29  
 Новый элемент библиотеки..... A-161  
 Номер версии..... A-125  
 Номер подчиненного ..... A-37  
 Номер ссылки ..... D-503  
 Нормальный стиль..... A-76  
 Область действия..... A-86, A-88  
 Общие ..... A-171  
 Общий..... D-499  
 Объявление ..... A-30, A-86  
 Окно вывода ..... A-85  
 Операнд (IL)..... D-502  
 Оператор ..... D-504  
 Операторы (IL) ..... B-277  
 Операция (IL)..... D-502  
 Описатель проекта..... A-34  
 Оптимизатор..... A-112  
 Опции компилятора.... A-34, A-111, A-145  
 Освобождение перехода ..... D-499  
 Открыть программу..... A-30, A-122  
 Открыть проект ..... A-23  
 Открыть файл ..... B-367  
 Отладчик..... A-36, A-123, A-147  
 Отрицательный контакт ..... B-250  
 Отслеживание изменений..... A-76  
 Ошибка ..... A-114  
 Ошибка выполнения..... A-127  
 Ошибка исполнения..... D-503  
 Ошибка исполнения..... A-33  
 Память ..... A-13  
 Панель управления ..... A-123  
 Параметер..... A-31  
 Параметр..... A-168  
 Параметр (С функции)..... C-439  
 Параметр (С функция)..... D-502  
 Параметр (Плата В/В) ..... D-502  
 Параметр (функциональный блок) .. C-449  
 Параметр платы..... A-102, A-166  
 Пароль.. A-24, A-105, A-162, A-178, A-180  
 Перейти..... A-45, A-56, A-78  
 Перекрестная ссылка ..... D-499  
 Перекрестные ссылки..... A-34, A-121  
 Переменная A-30, A-46, A-65, A-71, A-75, A-78, A-86, A-121, A-122, A-127, A-168, D-505  
 Переменная В/В..... D-501  
 Переменные ВВ..... C-432, C-433  
 Перенумеровать..... A-45  
 Переход..... A-38, A-128, D-504  
 Печатать программу..... A-82  
 Печать..... A-24, A-35, A-89, A-176  
 Печать..... A-174  
 Плата ..... A-100, A-101  
 Плата В/В ..... A-165, D-501  
 Подпрограмма ..... B-236  
 Подпрограмма ..... A-28, B-200, D-504  
 Подпрограмма FC ..... A-28  
 Подсмотреть переменную ..... A-134  
 Положительный контакт ..... B-250  
 Порядок выполнения ..... A-74  
 Последний шаг ..... D-500  
 Последовательная Функциональная  
 Диаграмма..... D-503  
 Последовательный..... B-198  
 Последовательный канал..... A-37  
 Поток ..... A-54, B-235, B-237, B-239  
 Поточковая диаграмма ..... A-51  
 Поточковая Диаграмма ..... D-500  
 Поточковые диаграммы ..... B-234  
 Пошаговый режим..... D-499  
 Пошаговый режим..... A-33  
 Право доступа..... A-178  
 Преобразование ..... A-107, D-499  
 Преобразование ASCII -> символа .. B-354  
 Преобразование символа -> ASCII .. B-353  
 Привод архива ..... A-172  
 Привязка В/В..... A-34  
**Присвоение (в ST,**  
 =) ..... **B-265**  
 Проверить..... A-33, A-110, A-167  
 Проверка четности чет/нечет..... B-350  
 Программа .. A-27, A-80, A-151, B-198, D-503

Программа верхнего уровня.....	D-504	Сетевой адрес.....	A-88, A-90
Проект.....	A-171, D-503	Сетевые адреса.....	A-94
Прожектор.....	A-138	Сетка.....	A-63
Прыжок.....	A-61, A-71, B-242, B-257	Сжатие.....	A-172
Прыжок на шаг.....	D-501	Симулятор.....	A-35, A-150, A-152
Прыжок на шаг.....	A-40	Симуляция.....	A-148
Псевдоним.....	A-67	Синус.....	B-340
Рабочее пространство отладки....	A-36	Скорость передачи.....	A-37
Разблокировать.....	A-126	Скрипт.....	A-152, A-154
Разделители проектов.....	A-22	Словарь..	A-30, A-86, A-122, A-168, C-433, C-447, D-500
Разделитель.....	D-503	Сложное оборудование В/В.....	A-165
Разединить.....	A-140	Слот.....	A-101, A-104
Расхождение.....	A-39, A-43	Случайное число.....	B-351
Реальная плата.....	D-503	Содержание.....	A-174
Реальное время.....	A-33, A-125	Соединение.....	A-71, A-73, A-74
Регистр (IL).....	D-503	СоединениеВ/В.....	D-501
Редактировать дескриптор проекта. A-23		Соединения В/В.....	A-100
Редактор FBD.....	A-69, A-80	Соединитель.....	A-54, B-237, D-499
Редактор FC.....	A-51	Создание массива.....	B-364
Редактор IL.....	A-80	Создать.....	A-33, A-110
Редактор LD.....	A-60	Сообщение.....	A-135, D-502
Редактор Quick LD.....	A-80	Сообщение компилятора.....	A-114
Редактор SFC.....	A-38, A-80	Сообщение об ошибке.....	A-85
Редактор ST.....	A-80	Сортировать.....	A-90
Редактор Поточковых Диграмм....	A-51	Сохранить.....	A-25
Режим реального времени.....	D-503	Сохранить список.....	A-134
Релейная Диаграмма.....	D-501	Специфический В/В.....	B-235
Ресурс.....	A-115	Список переменных..	A-134, A-136, A-141
Ресурсы.....	A-34	Список проектов.....	A-22, A-24
Решение.....	A-51, A-55, B-235, D-500	Спмсок Инструкций.....	D-501
Родительская программа.....	D-503	Старт.....	A-124
Родительская программа SFC... D-500		Стиль.....	A-76, A-141
Связь... A-37, A-71, A-73, A-74, A-126, A-144, A-186, B-235, B-237, B-239		Стоп.....	A-124
Связь FC.....	A-54	Страница.....	A-176
Сдвиг влево.....	B-343	Строка.....	D-504
Сдвиг вправо.....	B-344	Структурный текст.....	D-504
Сдвинуть проект.....	A-22	Ступень.....	A-66, A-73
Секция.....	A-27, D-503	Схождение.....	A-39, A-43
Секция Begin.....	D-498	Таблица преобразований.....	A-107
Секция End.....	D-500	Таблица преобразования.....	D-499
Секция Sequential.....	D-503	Таблица преобразования.....	A-109
Сетевой адрес.....	D-502	Таблица символов.....	A-189

Таймаут.....	A-37	Функциональная Блочная Диаграмма .....	D-500
Таймер .....	D-504	Функциональный блок.....	A-27, D-500
Таймирование цикла.....	A-151	Функциональный блок..	A-31, A-61, A-70, A-75, A-87, A-91, A-163, B-201, C-446
Текст .....	A-138	Функция..	A-27, A-31, A-163, A-167, B-200
Текущий результат (IL).....	D-499	Функция преобразования.....	A-170
Тест..	A-51, A-55, A-123, A-148, B-235, D-504	Целевая задача.....	A-111
Технические замечания .....	D-504	Целевой цикл.....	D-504
Технические замечания ..	A-102, C-432	Целое .....	D-501
Техническое замечание .....	A-163	Цель .....	D-504
Тип ..	A-86, A-88, A-100, A-121, A-166, D-504	Цикл.....	B-198, B-203
Тип витка .....	A-66	Цикл за Циклом.....	A-125
Тип контакта.....	A-66	Цикл профилера .....	A-151
Тип платы .....	A-101	Циклический.....	B-198, D-499
Точка .....	A-107, A-108	Четность .....	A-37
Точка останова....	A-126, A-128, D-498	Чтение массива.....	B-365
Угол .....	A-72	Чтение файла.....	B-371
Удаленный стиль.....	A-76	Чтение файла.....	B-376
Удалить FBD .....	A-74	Шаг .....	A-38, A-128, D-504
Удалить FC .....	A-56	Шина.....	A-60, A-61
Удалить LD.....	A-66	Шина питания.....	D-503
Удалить SFC .....	A-44	Шина питания.....	A-60, A-61, A-69
Удалить плату.....	A-101	Шпион.....	A-134, A-136, A-138
Удалить подстроку.....	B-355	Шрифт .....	A-177
Удалить программу .....	A-32	Экземпляр.....	A-87, A-91
Удалить текст .....	A-78	Экземпляр функционального блока C-447	
Уровень 2 SFC .....	D-501	Экспорт .....	A-95
Уровень защиты .....	A-178	Экспортировать функциональный блок .....	A-33
Условие .....	B-235	Экспортировать функцию .....	A-33
Условие (для перехода) .....	D-499	Язык.....	A-28, B-203
Файл		Язык C .....	C-431
определение конца файла.....	B-369	Язык C C-431, C-434, C-435, C-441, C-452, C-465, D-498	
Файл архива.....	A-173	Ячейка.....	D-498
Файл определения ресурсов .....	A-115		
Фоновая картинка .....	A-138		
Фронт.....	D-500		
Фронт контакта.....	B-250		
Функции преобразованияC-431, C-432			

