SoMachine

Руководство по программированию

11/2016

EIO0000000067.13

Schneider Electric
www.schneider-electric.com

Информация, представленная в этом документе, содержит общие определения и/или технические характеристики функционирования оборудования, упоминаемого в нём. Эта документация не предназначена и не должна использоваться для определения пригодности или надежности этих продуктов для конкретных пользовательских применений. Обязанность конечного пользователя или системного интегратора - выполнить соответствующий и полный анализ степени риска, оценку и тестирование продуктов для соответствующего применения или использования. Ни Schneider Electric, ни любой из его филиалов не несут ответственность за неправильное употребление содержащейся в документе информации. Пожалуйста, уведомьте нас, если Вы имеете какие-либо предложения по его улучшению или поправкам или нашли ошибки в этой публикации.

Никакая часть этого документа не может быть воспроизведена в любой форме или любыми средствами, электронными или механическими, включая фотокопирование, ксерокопирование или иные средства копирования или сохранения информации, без явно выраженного письменного разрешения Schneider Electric.

Все национальные, региональные и местные правила техники безопасности и стандарты должны соблюдаться при монтаже и использовании этого продукта. По соображениям безопасности и с целью гарантирования соответствия документированным данным системы только изготовитель должен выполнять ремонт компонентов.

При использовании устройства в применениях с техническими требованиями по технике безопасности, необходимо следовать соответствующим инструкциям.

Отказ от использования программного обеспечения Schneider Electric или одобренного им программного обеспечения с нашими аппаратными продуктами может привести к травмам, повреждениям или неверным результатам работы.

Несоблюдение этих указаний может привести к повреждению оборудования или травмам.

© 2016 Schneider Electric. Все права защищены.

Содержание

	Информация по безопасности	**17**
	Информация о данном руководстве	**19**
Часть I	**Введение**	**25**
Глава 1	**Общее введение в SoMachine Logic Builder**	**27**
	Что такое SoMachine Logic Builder?	**28**
	Задачи, выполняемые SoMachine Logic Builder	**29**
Глава 2	**Интерфейс пользователя SoMachine Logic Builder**	**31**
	Элементы экрана SoMachine Logic Builder	**32**
	Навигаторы с вкладками	**37**
	Вид каталога, снабженного вкладками	**43**
	Настройка интерфейса пользователя	**45**
	Интерфейс пользователя в режиме онлайн	**49**
	Меню и команды	**51**
Глава 3	**Фундаментальные понятия**	**53**
	Введение и фундаментальные понятия	**53**
Часть II	**Конфигурация**	**55**
Глава 4	**Установка устройств**	**57**
	Интеграция Sercos устройств от сторонних поставщиков	**57**
Глава 5	**Управление устройствами**	**59**
5.1	Добавление устройств перетаскиванием (Drag and Drop)	**60**
	Добавление устройств перетаскиванием (Drag and Drop)	**60**
5.2	Добавление устройств из контекстного меню или кнопкой Plus (плюс)	**63**
	Добавление контроллера	**64**
	Добавление устройств расширения	**65**
	Добавление менеджеров коммуникаций	**66**
	Добавление устройств к менеджеру коммуникации	**68**
	Добавление устройств из шаблона	**70**
5.3	Обновление устройств	**71**
	Обновление устройств	**71**
5.4	Преобразование устройств	**73**
	Преобразование устройств	**73**
5.5	Преобразование проектов	**77**
	Преобразование проектов SoMachine Basic и Twido	**77**

Глава 6 Диалоговые окна общих редакторов устройств 93
6.1 Конфигурация устройств 94
Общая информация о редакторах устройств. 95
Выбор контроллера 97
Коммуникационные параметры настройки 111
Конфигурация 115
Приложения 116
Файлы 118
Регистрация 119
Параметры настройки ПЛК 121
Пользователи и группы 123
Task Deployment – развёртывание задач 133
Статус 134
Информация 134
6.2 I/O Mapping – Разметка Вх/Вых 135
I/O Mapping – Разметка Вх/Вых 136
Работа с диалогом I/O Mapping. 138
I/O Mapping в онлайн режиме 141
Неявные переменные для форсированной установки Вх/Вых 142
Часть III Программа 145
Глава 7 Компоненты программы 147
7.1 Program Organization Unit (POU) 148
POU 149
Добавление и вызов POUs 150
Программа 154
Функция 156
Метод 158
Свойство 161
Интерфейс 163
Свойство интерфейса. 167
Действие 170
Внешние функция, функциональный блок, метод 172
POUs для неявных проверок 173
7.2 Функциональный блок 174
Общая информация 175
Экземпляр функционального блока 177
Вызов функционального блока 178
Расширение функционального блока 180

Реализация интерфейсов 183
Вызов метода 185
Указатель SUPER. 187
Указатель THIS. 189
7.3 Объекты приложения 192
Data Type Unit (DUT) – Тип единицы данных 193
Global Variable List (GVL) – Список глобальных переменных 195
Global Network Variable List (GNVL) – Список сетевых глобальных переменных 197
Persistent Variables - Перманентные переменные. 205
Внешний файл. 207
Text List - Список текстов 209
Пул изображений 216
7.4 Приложения 218
Приложения 218
Глава 8 Конфигурация задач 219
Конфигурация задач 220
Добавление задач. 221
Глава 9 Управление приложениями 223
9.1 Общая информация. 224
Введение 224
9.2 Построение и загрузка приложений. 226
Построение приложений 227
Логин 228
Процесс построения при изменённых Applications. 231
Загрузка приложения 232
9.3 Выполнение приложений. 242
Выполнение приложений. 242
9.4 Обслуживание приложений 243
Мониторинг 244
Отладка 245
Часть IV Логические редакторы 247
Глава 10 Редактор FBD/LD/IL. 249
10.1 Информация о редакторе FBD/LD/IL 250
Редактор FBD/LD/IL. 251
Язык Function Block Diagram (FBD) - диаграмм функциональных блоков 252
Язык Ladder Diagram (LD) - лестничных диаграмм 253
Язык Instruction List (IL) – списка инструкций 254
Модификаторы и операторы в IL 256

Работа в редакторе FBD и LD 258
Работа в IL редакторе 262
Позиции курсора в FBD, LD и IL 268
Меню FBD/LD/IL. 272
Редактор FBD/LD/IL в режиме онлайн. 273
10.2 Элементы FBD/LD/IL 279
Панель инструментов FBD/LD/IL. 280
Цепь в FBD/LD/IL 282
Назначение в FBD/LD/IL 284
Переход в FBD/LD/IL. 284
Метки в FBD/LD/IL. 284
Блоки в FBD/LD/IL 285
Инструкция `RETURN` в FBD/LD/IL 286
Ветвление / Подвешивание катушек в FBD/LD/IL. 287
Параллельные ветви. 290
Set/Reset в FBD/LD/IL 292
Катушки Set/Reset. 293
10.3 Элементы LD. 294
Контакт. 295
Катушка 296
Глава 11 Редактор Continuous Function Chart (CFC) 297
Язык Continuous Function Chart (CFC). 298
Редактор CFC 299
Позиции курсора в CFC. 301
CFC элементы / Панель инструментов 303
Работа в CFC редакторе 309
Редактор CFC в онлайн режиме 312
Ориентированный на страницы CFC редактор 314
Глава 12 Редактор Sequential Function Chart (SFC) 317
Редактор SFC. 318
Язык SFC - Sequential Function Chart 320
Позиции курсора в SFC. 321
Работа в SFC редакторе. 323
Свойства SFC элемента 325
SFC элементы / Панель инструментов. 327

Спецификаторы для действий в SFC. 338
Неявные переменные - SFC флаги 339
Последовательность выполнения в SFC 344
SFC редактор в онлайн режиме. 346
Глава 13 Редактор Structured Text (ST). 347
13.1 Информация о ST редакторе. 348
ST редактор 349
ST редактор в онлайн режиме 350
13.2 Язык Structured Text ST / Extended Structured Text (ExST) 354
Структурированный текст ST / Расширенный структурированный текст ExST 355
Выражения 356
Инструкции 358
Часть V Редакторы объектов 369
Глава 14 Редакторы объявлений 371
Текстовый редактор объявлений 372
Табличный редактор объявления. 373
Редактор декларации в режиме онлайн. 376
Глава 15 Редактор Device Type Manager (DTM) 377
Редактор DTM 377
Глава 16 Редактор Data Unit Type (DUT). 379
Редактор Data Unit Type 379
Глава 17 Редактор Global Variables List (GVL). 381
Редактор GVL 381
Глава 18 Редактор Network Variables List (NVL 383
18.1 Информация о редакторе NVL. 384
Редактор списка сетевых переменных 384
18.2 Общая информация о сетевых переменных 385
Введение в список сетевых переменных (NVL). 386
Конфигурирование обмена сетевыми переменными 389
Правила списка сетевых переменных Network Variables List (NVL) . . 393
Состояние работы отправителя и получателя. 395
Пример. 396
Совместимость. 401

Глава 19	**Редактор задач**	**405**
	Информация о конфигурации задачи	406
	Вкладка свойств	407
	Вкладка монитора	408
	Конфигурация определенной задачи	410
	Обработка задачи в режиме онлайн	414
Глава 20	**Редактор списка наблюдения**	**415**
	Вид наблюдения / Редактор списка наблюдения	416
	Создание списка наблюдения	417
	Список наблюдения в режиме онлайн	419
Глава 21	**Инструменты в логических редакторах**	**421**
	Средство поиска функции и функционального блока	422
	Ассистент ввода	425
Часть VI	**Инструменты**	**427**
Глава 22	**Регистрация данных**	**429**
	Введение в регистрацию данных	429
Глава 23	**Менеджер рецептов**	**431**
	Менеджер рецептов	432
	Определение рецепта	435
	Команды `RecipeMan`	439
Глава 24	**Редактор трассировки**	**449**
24.1	Объект трассировки	450
	Основы трассировки	451
	Создание объекта трассировки	453
24.2	Конфигурация трассировки	456
	Параметры настройки переменных	457
	Параметры настройки записи	460
	Расширенные параметры настройки трассировки	463
	Редактирование изображения	465
	Изображение оси Y	469
24.3	Редактор трассировки в режиме онлайн	470
	Редактор трассировки в режиме онлайн	470
24.4	Действия клавиатуры для графиков трассировки	471
	Быстрые команды клавиатуры	471
Глава 25	**Редактор конфигурации символов**	**473**
	Редактор конфигурации символов	474
	Конфигурация символов	477
	Добавление конфигурации символов	478

Глава 26 Обмен данных SoMachine контроллер – HMI 481
Определение одной переменной SoMachine 482
Опубликование переменных в контроллерной части 486
Выбор переменных в HMI части. 488
Публикация переменных в HMI части 489
Параметризация физической среды передачи 491
Часть VII Справка по программированию. 493
Глава 27 Объявление переменных 495
27.1 Объявление 496
Общая информация. 497
Рекомендации по именованию идентификаторов 500
Инициализация переменных 505
Объявление 506
Режим ярлыка 507
`AT` объявление 508
Ключевые слова. 509
27.2 Типы переменных. 513
Типы переменных. 514
Ключевые слова атрибута для типов переменных 518
Конфигурация переменных - `VAR_CONFIG` 522
27.3 Типы методов 524
Методы `FB_init`, `FB_reinit`. 525
Метод `FB_exit`. 528
27.4 Инструкции прагмы. 529
Инструкции прагмы. 530
Прагмы сообщения. 532
Условные прагмы 534
27.5 Атрибуты прагм 542
Атрибуты прагм. 543
Задаваемые пользователем атрибуты 544
Атрибут `call_after_init`. 546
Атрибут `displaymode`. 547
Атрибут `ExpandFully`. 548
Атрибут `global_init_slot`. 549
Атрибут `hide` 550
Атрибут `hide_all_locals`. 551
Атрибут `initialize_on_call` 552
Атрибут `init_namespace`. 552

Атрибут init_On_Onlchange 553
Атрибут instance-path 554
Атрибут linkalways 555
Атрибут monitoring 556
Атрибут namespace 560
Атрибут no_check 562
Атрибут no_copy 562
Атрибут no-exit 563
Атрибут no_init 564
Атрибут no_virtual_actions 565
Атрибут obsolete 568
Атрибут pack_mode 569
Атрибут qualified_only 570
Атрибут reflection 570
Атрибут subsequent 571
Атрибут symbol 572
Атрибут warning disable 574
27.6 Функциональность умного кодирования 575
Умное кодирование 575
Глава 28 Типы данных 577
28.1 Общая информация 578
Типы данных 578
28.2 Стандартные типы данных 579
Стандартные типы данных 579
28.3 Расширения к стандарту IEC 582
UNION 583
LTIME 583
WSTRING 584
BIT 584
References 585
Указатели 587
28.4 Типы задаваемых пользователем данных 590
Типы задаваемых пользователем данных 591
Множества 592
Структуры 595
Перечисления 597
Типы поддиапазона 599

Глава 29 Программирование инструкций 603
29.1 Соглашение о присвоении имен. 604
Общая информация 604
29.2 Префиксы 606
Префиксные части. 607
Порядок префиксов 608
Префикс объема 610
Префикс типа данных. 611
Property Prefix. 613
Префикс POU. 614
Префикс пространства имен 615
Глава 30 Операторы 617
30.1 Арифметические операторы 618
ADD 619
MUL 621
SUB 623
DIV 625
MOD 628
MOVE. 630
SIZEOF. 631
30.2 Операторы строки битов. 632
AND 633
OR. 634
XOR 635
NOT 636
30.3 Операторы сдвига разряда 637
SHL 638
SHR 640
ROL 641
ROR 643
30.4 Операторы выбора. 645
SEL 646
MAX 647
MIN 648
LIMIT. 649
MUX 650

30.5 Операторы сравнения 651
GT. 652
LT 653
LE 654
GE. 655
EQ. 656
NE 657
30.6 Операторы адреса. 658
ADR. 659
Оператор содержания. 660
BITADR 661
30.7 Оператор вызова 662
CAL 662
30.8 Операторы преобразования типов. 663
Функции преобразования типов. 664
Преобразования BOOL_TO. 665
Преобразования TO_BOOL. 667
Преобразование между типами целого числа. 669
Преобразования REAL_TO / LREAL_TO. 670
Преобразования TIME_TO/TIME_OF_DAY. 672
Преобразования DATE_TO/DT_TO 674
Преобразования STRING_TO. 676
TRUNC 678
TRUNC_INT. 679
Преобразования ANY_. . . _TO. 679
30.9 Числовые функции 680
ABS. 681
SQRT. 682
LN. 683
LOG. 684
EXP. 685
SIN. 686
COS. 687
TAN. 688
ASIN. 689
ACOS. 690
ATAN. 691
EXPT. 692

30.10 Операторы расширения IEC 693
Операторы расширения IEC 694
__DELETE 695
__ISVALIDREF 698
__NEW 699
__QUERYINTERFACE 702
__QUERYPOINTER 704
Операторы объема 706
30.11 Оператор инициализации 708
Оператор INI 708
Глава 31 Операнды 709
31.1 Константы 710
Константы BOOL 711
Константы TIME 712
Константы DATE 714
Константы DATE_AND_TIME 715
Константы TIME_OF_DAY 716
Константы Number 717
Константы REAL/LREAL 718
Константы STRING 719
Введенные константы / Введенные литералы 720
31.2 Переменные 721
Переменные 722
Адресация битов в переменных 723
31.3 Адреса 725
Адреса 725
31.4 Функции 728
Функции 728
Часть VIII Шаблоны SoMachine 729
Глава 32 Общая информация о шаблонах SoMachine 731
32.1 Шаблоны SoMachine 732
Общая информация о шаблонах SoMachine 733
Администрация шаблонов SoMachine 736

Глава 33 Управление шаблонами устройства 743
33.1 Управление шаблонами устройства 744
Факты шаблонов устройства 745
Добавление устройств из шаблона 746
Создание шаблона устройства на основе полевых устройств или модулей ввода-вывода. 749
Визуализация, подходящая для создания шаблонов устройства 750
Дополнительная информация об интегрировании управляющей логики в шаблоны устройства. 751
Шаги, чтобы создать шаблон устройства 753
Глава 34 Управление шаблонами функций. 757
34.1 Управление шаблонами функций. 758
Факты шаблонов функций 759
Добавление функций из шаблона 760
Прикладные функции как основание для шаблонов функций. 766
Шаги, чтобы создать шаблон функции 768
Часть IX Поиск и устранение неисправностей и FAQ . . 773
Глава 35 Общая информация - поиск и устранение неисправностей и FAQ. 775
35.1 Часто задаваемые вопросы 776
Как я могу включить и сконфигурировать аналоговые входы на CANopen? 777
Почему выполнение запуска SoMachine иногда более медленное? . 778
Как я могу управлять ярлыками и меню? 779
Как я могу увеличить предел памяти, доступный для SoMachine в 32-разрядных операционных системах? 781
Как я могу уменьшить потребление памяти SoMachine? 782
Как я могу увеличить производительность времени изготовления SoMachine? 782
Что я могу сделать в случае проблем с Modbus IOScanner на последовательной линии? 783
Что я могу сделать, если моя коммуникация Network Variables List (NVL) была приостановлена? 784
Что я могу сделать, если многократная загрузка неуспешна на контроллере HMI? 784
Глава 36 Доступ к контроллерам - поиск и устранение неисправностей и FAQ 785
36.1 Поиск и устранение неисправностей: доступ к новым контроллерам 786
Доступ к новым контроллерам 787
Соединение через IP-адрес и Адресную информацию. 789
36.2 FAQ - что я могу сделать в случае проблем соединения с контроллером? 791
FAQ - Почему соединение с контроллером невозможно? 792
FAQ - Почему имеющаяся связь между PC и контроллером прервана? 795

Приложения 797
Приложение A Сетевые коммуникации. 799
Топология сети 800
Адресация и маршрутизация 801
Структура адресов 803
Приложение B Использование сервера OPC 807
Общая информация 808
Декларирование переменной, используемой с OPC. 810
Конфигурация сервера OPC. 813
Использование сервера CoDeSys OPC. 819
Приложение C Язык сценария. 821
C.1 Общая информация. 822
Введение 823
Выполнение сценариев. 826
Наиболее успешная практика. 829
Чтение документации API.NET 830
Точки входа 832
C.2 Примеры механизма выполнения сценария Schneider Electric 833
Параметры устройства 834
Версия компилятора 836
Профиль визуализации 837
Библиотеки обновления 838
Уберите и создайте приложение. 839
Коммуникационные параметры настройки 840
Сообщения диагностики сброса. 840
Перезагрузите контроллер 841
Преобразуйте устройство 842
Сравнение проектов. 845
Усовершенствованные функции управления библиотеки 846
Доступ к POUs 847
C.3 Примеры механизма выполнения сценария CoDeSys. 848
Проект 849
Онлайн-приложение 854
Объекты 857
Устройства 858
Система / Пользовательский интерфейс (UI) 860

Чтение значений 862
Чтение значений из рецепта и посылка электронного письма. 863
Определение дерева устройства открытого проекта 865
Пример сценария 4: Импорт устройства в PLCOpenXML Subversion. 866
Приложение D Управление пользователями для Soft PLC 869
Общая информация об управлении пользователями для Soft PLC . 870
Пользователи и группы 872
Права доступа. 875
Приложение E Наборы функций контроллера для миграции 879
Наборы функций контроллера для миграции 879
Словарь терминов 883
Указатель 887

Информация по безопасности

Важная информация

ЗАМЕЧАНИЕ

Внимательно прочитайте данные правила и ознакомьтесь с оборудованием, чтобы убедиться в его идентичности устройству, прежде чем осуществлять его установку, эксплуатацию или техническое обслуживание. Приведённые ниже специальные сообщения, которые могут находиться в документации или на оборудовании, имеют целью предупреждение о потенциальных опасностях или привлечение внимания к информации, призванной разъяснить или упростить ту или иную процедуру.

Этот символ в комбинации с предупреждающей табличкой «Danger» (Опасно) или «Warning» (Предупреждение) сигнализирует о присутствии опасности поражения электрическим током, из-за которой несоблюдение правил техники безопасности может привести к телесным повреждениям.

Этот символ обозначает опасность. Он используется, чтобы предупредить Вас о потенциальном риске получения телесных повреждений. Во избежание травм или летального исхода неукоснительно соблюдайте правила безопасности, указанные рядом с этим символом.

DANGER

Надписью "**DANGER**" (ОПАСНО) обозначают чрезвычайно опасные ситуации, которые, если их не избежать, приведут к смерти, серьезным травмам или выходу оборудования из строя.

WARNING

Надписью "**Warning**" (ПРЕДУПРЕЖДЕНИЕ) обозначают возможные опасные ситуации, которые, если их не избежать, могут привести к смерти, серьезным травмам или выходу оборудования из строя.

CAUTION

Надписью "**CAUTION**" (ВНИМАНИЕ) обозначают возможные опасные ситуации, которые, если их не избежать, могут привести к травмам или выходу оборудования из строя.

NOTICE

Надписью "***NOTICE***" (ЗАМЕЧАНИЕ) обозначают ситуации, которые не связаны с физическими повреждениями.

ВАЖНОЕ ЗАМЕЧАНИЕ

Электрическое оборудование должно быть установлено, введено в эксплуатацию, обслуживаться только квалифицированным персоналом. Компания Schneider Electric не несет ответственности за любые последствия в результате использования данного материала.

Квалифицированным специалистом является тот, кто имеет знания по конструкции и навыки по эксплуатации электрооборудования и его установке и прошёл обучение по технике безопасности для распознавания и предотвращения имеющихся опасностей.

Информация о данном руководстве

Краткий обзор

Цель данного руководства

Настоящее Руководство содержит описание графического интерфейса пользователя для программного обеспечения SoMachine и обеспечиваемых им функций. За дополнительной информацией обращайтесь к отдельным документам, обеспечиваемым в онлайн справочной системе SoMachine.

Замечания по применимости

Этот документ был обновлен для версии SoMachine V4.2.

Дополнительная документация

Наименование документа	Каталожный номер
SoMachine Introduction (Введение в SoMachine)	EIO0000000734 (ENG); EIO0000000787 (FRE); EIO0000000788 (GER); EIO0000000790 (SPA); EIO0000000789 (ITA); EIO0000000791 (CHS)
SoMachine Menu Commands Online Help (Онлайн-справка команд меню SoMachine)	EIO0000001854 (ENG); EIO0000001855 (FRE); EIO0000001856 (GER); EIO0000001858 (SPA); EIO0000001857 (ITA); EIO0000001859 (CHS)
SoMachine Central User Guide (Руководство пользователя по Central SoMachine)	EIO0000001659 (ENG); EIO0000001660 (FRE); EIO0000001661 (GER); EIO0000001663 (SPA); EIO0000001662 (ITA); EIO0000001664 (CHS)
SoMachine Compatibility and Migration User Guide (Руководство пользователя по совместимости и миграции SoMachine)	EIO0000001684 (ENG); EIO0000001685 (FRE); EIO0000001686 (GER); EIO0000001688 (SPA); EIO0000001687 (ITA); EIO0000001689 (CHS)

Наименование документа	Каталожный номер
SoMachine Functions and Libraries User Guide (Руководство пользователя по функциям и библиотекам SoMachine)	EIO0000000735 (ENG); EIO0000000792 (FRE); EIO0000000793 (GER); EIO0000000795 (SPA); EIO0000000794 (ITA); EIO0000000796 (CHS)
SoMachine Controller Assistant User Guide (Руководство пользователя по Ассистенту контроллера в SoMachine)	EIO0000001671 (ENG); EIO0000001672 (FRE); EIO0000001673 (GER); EIO0000001675 (SPA); EIO0000001674 (ITA); EIO0000001676 (CHS)
Modicon M238 Logic Controller Programming Guide (Руководство по программированию логического контроллера Modicon M238)	EIO0000000384 (ENG); EIO0000000385 (FRE); EIO0000000386 (GER); EIO0000000388 (SPA); EIO0000000387 (ITA); EIO0000000389 (CHS)
SoMachine Device Type Manager (DTM) User Guide (Руководство пользователя по менеджеру типов устройств (DTM - Device Type Manager) SoMachine)	EIO0000000673 (ENG); EIO0000000674 (FRE); EIO0000000675 (GER); EIO0000000676 (SPA); EIO0000000677 (ITA); EIO0000000678 (CHS)
Twido Emulation Support Library Guide (Руководство по вспомогательной библиотеке эмуляции Twido)	EIO0000001692 (ENG); EIO0000001693 (FRE); EIO0000001694 (GER); EIO0000001696 (SPA); EIO0000001695 (ITA); EIO0000001697 (CHS)
SoMachine Network Variable Configuration SE_NetVarUdp Library Guide (Руководство по библиотеке SE_NetVarUdp конфигурации сетевых переменных в SoMachine)	EIO0000001151 (ENG); EIO0000001152 (FRE); EIO0000001153 (GER); EIO0000001155 (SPA); EIO0000001154 (ITA); EIO0000001156 (CHS)

Вы можете загрузить эти технические публикации и другую техническую информацию с нашего Web-сайта по адресу: http://download.schneider-electric.com

Информация, связанная с изделием

ПРЕДУПРЕЖДЕНИЕ

ПОТЕРЯ УПРАВЛЕНИЯ

- Проектировщик любой схемы управления должен рассмотреть потенциальные режимы отказа цепей управления и, для определенных критических функций управления, обеспечить средства достижения безопасного состояния во время и после отказа цепи. Примеры критических функций управления - аварийная остановка и остановка при выходе за допустимые границы перемещения (конечные выключатели), перебой в питании и перезапуск.
- Раздельные или избыточные цепи управления должны быть обеспечены для критических функций управления.
- Системные цепи управления могут включать коммуникационные каналы связи. Внимание должно быть уделено последствиям непредвиденных задержек в передаче или ошибкам канала связи.
- Соблюдайте все инструкции по технике безопасности и локальные руководства по безопасности.[1]
- Каждая реализация этого оборудования должна быть индивидуально и полностью протестирована на правильное функционирование прежде чем быть введённой в эксплуатацию.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

[1] Для получения дополнительной информации обратитесь к NEMA ICS 1.1 (последняя редакция) "Safety Guidelines for the Application, Installation, and Maintenance of Solid State Control " и в NEMA ICS 7.1 (последняя редакция) "Safety Standards for Construction and Guide for Selection, Installation and Operation of Adjustable-Speed Drive Systems" или их эквиваленты руководств для Вашего конкретного расположения.

ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННАЯ РАБОТА ОБОРУДОВАНИЯ

- Используйте только программное обеспечение, утвержденное Schneider Electric для использования с этим оборудованием.
- Обновляйте свою прикладную программу каждый раз, когда Вы изменяете физическую аппаратную конфигурацию.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

Терминология, используемая в стандартах

Технические термины, терминология, символы и соответствующие описания в этом руководстве, или которые появляются в или на самих продуктах, обычно получены из терминов или определений международных стандартов.

В области функциональных систем безопасности, приводов и общей автоматизации, это может включать, но не ограничено, терминами, такими как *безопасность, функция безопасности, безопасное состояние, отказ, сброс отказа, неправильное функционирование, ошибка, сообщение об ошибке, опасное,* и т.д.

Среди других эти стандарты включают:

Стандарт	Описание
EN 61131-2:2007	(Программируемые контроллеры, часть 2: Требования к оборудованию и тесты.

Стандарт	Описание
ISO 13849-1:2008	Безопасность машинного оборудования: Части систем управления, связанные с безопасностью. Общие принципы дизайна.
EN 61496-1:2013	Безопасность машинного оборудования: Электрическое защитное оборудование. Часть 1: Общие требования и тесты.
ISO 12100:2010	Безопасность машинного оборудования - Общие принципы дизайна - оценка степени риска и снижение риска
EN 60204-1:2006	Безопасность машинного оборудования - Электрооборудование машин - Часть 1: Общие требования
EN 1088:2008 ISO 14119:2013	Безопасность машинного оборудования - Взаимосвязанные устройства связи с защитой - Принципы дизайна и выбора
ISO 13850:2006	Безопасность машинного оборудования - Аварийная остановка - Принципы дизайна
EN/IEC 62061:2005	Безопасность машинного оборудования - Функциональная безопасность связанных с безопасностью электрических, электронных, и электронных программируемых систем управления
IEC 61508-1:2010	Функциональная безопасность электрических/электронных/программируемых электронных связанных с безопасностью систем: Общие требования.
IEC 61508-2:2010	Функциональная безопасность электрических/электронных/программируемых электронных связанных с безопасностью систем: Требования для электрических/электронных/программируемых электронных связанных с безопасностью систем.
IEC 61508-3:2010	Функциональная безопасность электрических/электронных/программируемых электронных связанных с безопасностью систем: Требования к программному обеспечению.
IEC 61784-3:2008	Коммуникация цифровых данных для измерения и управления: Функционально безопасные полевые шины.
2006/42/EC	Директива машинного оборудования
2014/30/EU	Директива электромагнитной совместимости
2014/35/EU	Низковольтная директива

Кроме того, термины, используемые в существующем документе, могут мимоходом быть использованы, поскольку они получены из других стандартов, таких как:

Стандарт	Описание
IEC 60034 series	Вращающиеся электрические машины
IEC 61800 series	Системы электроприводов с регулируемой скоростью
IEC 61158 series	Коммуникации цифровых данных для измерения и управления – Полевая шина для использования в промышленных системах управления

Наконец, термин *зона работы (действия)* может быть использован в сочетании с описанием определенных опасностей и определен, как это сделано для *зоны опасности* или *опасной зоны* в *Директиве Машинного оборудования (2006/42/EC)* и *ISO 12100:2010*.

ПРИМЕЧАНИЕ: Вышеупомянутые стандарты могут применяться или могут не применяться к определенным продуктам, процитированным в настоящей документации. Для получения дополнительной информации относительно отдельных стандартов, применимых к продуктам, описанным здесь, посмотрите таблицы характеристик для ссылок на эти продукты.

Часть I

Введение

Содержание этой части

Эта часть содержит следующие главы:

Глава	Название главы	Страница
1	Общее введение в SoMachine Logic Builder	27
2	Интерфейс пользователя SoMachine Logic Builder	31
3	Базовые концепции	53

Глава 1

Общее введение в SoMachine Logic Builder

Содержание главы

Эта глава содержит следующие темы:

Тема	Страница
Что такое SoMachine Logic Builder?	28
Задачи, выполняемые SoMachine Logic Builder	29

Что такое SoMachine Logic Builder?

Общее описание

Logic Builder (Компоновщик логики) обеспечивает среду конфигурирования и программирования для проектов SoMachine, которые Вы создаете с SoMachine Central.

Он отображает различные элементы Вашего проекта в раздельных видах, которые Вы можете расположить в интерфейсе пользователя SoMachine и на Вашем рабочем столе согласно Вашим конкретным требованиям. Эта структура видов позволяет Вам добавлять аппаратные и программные элементы к своему проекту методом drag and drop (выделения и перемещения). Основные диалоговые окна конфигурации, позволяющие Вам создавать содержание проекта, расположены в центре экрана Logic Builder.

В дополнение к простой конфигурации и программированию, Logic Builder также обеспечивает мощные функции диагностики и обслуживания.

Задачи, выполняемые SoMachine Logic Builder

Конфигурирование и программирование проектов

Logic Builder позволяет Вам программировать логику и добавлять устройства к проектам SoMachine, которые Вы создаете с SoMachine Central.

Чтобы помочь Вам в выполнении этой задачи, он обеспечивает следующие функции:

- Отдельные виды для аппаратного каталога **Controller, HMI & iPC, Field Devices & Modules, Diverse**, позволяют Вам добавлять устройства к своему проекту простым перетаскиванием (drag and drop). Это также позволяет Вам использовать шаблоны устройств и шаблоны функций.
- Отдельные виды каталога программного обеспечения для **Variables, Assets, Macros, ToolBox, Libraries** позволяют Вам добавлять различные типы элементов программного обеспечения простым перетаскиванием (drag and drop). Вид **Assets**, например, позволяет Вам создавать и управлять вашими функциональными блоками и POUs (программными единицами).

Чтобы вывести на экран только соответствующие виды для задачи, которая выполняется, SoMachine обеспечивает отдельные перспективы *(см. стр. 47)* для аппаратной конфигурации, конфигурации программного обеспечения и режима онлайн. Вам разрешено адаптировать эти перспективы по умолчанию к Вашим индивидуальным требованиям и создать Ваши собственные перспективы с видами, которые Вы используете наиболее часто.

Построение проектов

Logic Builder обеспечивает различные способы (такие как **Build** (Построение), **Build all** (Построение всего) или **Clean all** (Очистка всего)) построения Вашего проекта SoMachine.

Коммуникация с контроллером

Logic Builder обеспечивает функции сканирования, чтобы обнаружить доступные контроллеры в сети Ethernet. Он поддерживает различные протоколы для связи с контроллером.

После того, как коммуникация была установлена, приложения могут быть загружены на или выгружены из контроллера. Приложения в контроллере могут быть запущены и остановлены.

Онлайн функции и мониторинг

Онлайн функции и функции мониторинга Logic Builder позволяют Вам выполнять следующие задачи:

- Онлайн мониторинг значений в коде программы и в видах **Watch**
- Выполнение онлайн изменений
- Онлайн конфигурация трассировок
- Наблюдение трассировок онлайн
- Взаимодействие с Вашей машиной при помощи встроенных визуализаций в режиме онлайн для целей диагностики и тестирования
- Чтение статуса (состояния) контроллеров и устройств
- Обнаружение потенциальных ошибок программирования логики при помощи функции отладки

Глава 2

Интерфейс пользователя SoMachine Logic Builder

Содержание главы

Эта глава содержит следующие темы:

Тема	Страница
Элементы экрана SoMachine Logic Builder	32
Навигаторы с вкладками	37
Вид **каталога,** снабженного вкладками	43
Настройка интерфейса пользователя	45
Интерфейс пользователя в онлайн режиме	49
Меню и команды	51

Элементы экрана SoMachine Logic Builder

Обзор

Logic Builder состоит из следующих элементов:

- Меню и панели инструментов
- Виды **навигатора**
- Виды **каталога**
- Основная область редактора

Когда Вы открываете Logic Builder, он обеспечивает вид экрана по умолчанию. Этот документ описывает позиции по умолчанию.

Вы можете адаптировать элементы согласно Вашим индивидуальным требованиям, как описано в главе *Настройка интерфейса пользователя* (*см. стр. 45*). Вы можете видеть и изменять текущие параметры настройки в диалоговом окне **Customize**. Оно по умолчанию доступно в меню **Tools**.

Вы можете также расположить виды и окна в любое время через смещение, прикрепление/расстыковку видов, изменение размеров или закрытие окон. Позиции сохраняются вместе с проектом. Когда Вы вновь открываете проект, элементы располагаются в позиции, где они были при сохранении проекта. Позиции видов сохраняются отдельно в перспективах *(см. стр. 47)*.

Экран Logic Builder по умолчанию

Позиции меню, панелей и видов на экране Logic Builder по умолчанию

1. Menu bar (строка меню)
2. Toolbar (панель инструментов)
3. Multi-tabbed Navigators (Навигаторы с вкладками): Devices tree (дерево устройств), Tools tree (дерево инструментов), Applications tree (дерево приложений)
4. Messages view (Виды сообщений)
5. Information and status bar (строка информации и состояния)
6. Multi-tabbed catalog view (виды каталога с вкладками): hardware catalog (аппаратный каталог): Controller (контроллер), HMI & iPC, Devices & Modules (устройства & модули), Diverse software catalog (разнообразные каталоги программного обеспечения): Variables (переменные), Assets (активы). Macros (макросы), ToolBox (окно инструментов), Libraries (библиотеки)
7. Multi-tabbed editor view (Виды редактора с вкладками)

Стандартные компоненты

Экран Logic Builder содержит следующие компоненты, которые видимы по умолчанию:

Компонента	Описание
Menu bar (Строка меню)	Предоставляет меню, которое содержит доступные команды, как это определено в **Tools** → диалоговом окне **Customize**.
Toolbar (Панель инструментов)	Содержит кнопки, запускающие доступные инструменты, как это определено в **Tools** → диалоговом окне **Customize**.
Multi-tabbed **Navigators** (**Навигаторы** с вкладками)	Следующие **Навигаторы** доступны как вкладки, где различные объекты проекта организованы в древовидной структуре: ● **Devices tree (дерево устройств)** ● **Applications tree (дерево приложений)** ● **Tools tree (дерево инструментов)** За дополнительной информацией обратитесь к главе *Навигаторы с вкладками* (*см. стр. 37*).
Messages view (Вид **сообщений)**	Обеспечивает сообщения при операциях precompile (предварительной компиляции), compile (компиляции), build (построения), download (загрузки). Обратитесь за деталями к описанию команд **Messages** view *(см. SoMachine, Menu Commands, Online Help*).
Information and status bar Строка информации и состояния	Обеспечивает следующую информацию: ● Информация о текущем пользователе ● Информация о режиме редактирования и текущей позиции, если редактор открыт. Для получения дополнительной информации смотрите раздел «*Строка информации и состояния*» в этой главе.
Multi-tabbed **Catalog** view Вид **каталога** с вкладками	Вид **Catalog (к**аталог) состоит из различных вкладок, где перечислены доступные аппаратные и программные объекты: ● **Hardware Catalog (Аппаратный каталог)** ❍ **Controller (Контроллер)** ❍ **HMI & iPC** ❍ **Devices & Modules (Устройства и модули)** ❍ **Diverse (Разнообразное)** ● **Software Catalog Программный каталог** ❍ **Variables (Переменные)** ❍ **Assets (Активы)** ❍ **Macros (Макросы)** ❍ **ToolBox (Окно инструментов)** ❍ **Libraries (Библиотеки)** Для получения дополнительной информации обратитесь к главе *Вид каталога с вкладками* (*см. стр. 43*).
Multi-tabbed editor window Окно редактора с вкладками	Используется для создания определенного объекта в соответствующем редакторе. В случае редакторов языка (например, редактора ST, редактора CFC), обычно окно комбинирует редактор языка в своей нижней части и редактора объявлений переменных в верхней части. В случае других редакторов это могут быть диалоговые окна (например, редактор задач, редактор устройств). Имя POU или объекта ресурса выведено на экран в строке заголовка этого вида. Вы можете открыть объекты в окне редактора в режиме офлайн или режиме онлайн, выполнив команду **Edit Object**.

Строка информации и состояния

Панель на нижней границе экрана Logic Builder обеспечивает 3 типа информации:

- Информация о зарегистрированном (logged-in) пользователе.
- Если Вы работаете в окне редактора: позиция курсора и состояние режима редактирования.
- В режиме онлайн: текущий статус программы.

Информация о зарегистрированном пользователе

У каждого проекта есть управление пользователями и управление доступом (*см. стр. 123*). Зарегистрированного в настоящее время пользователя называют в строке состояния.

Позиции курсора в окнах редактора

Позиция курсора отсчитывается от левого или верхнего поля окна редактора.

Аббревиатура	Описание
Ln	Строка (Line), в которую установлен курсор.
Col	Столбец (Column), в который установлен курсор. (Столбец содержит в себе ровно 1 пробел, символ или цифру.)
Ch	Число символов (characters). (В этом контексте символ может быть единственным символом или цифрой, а также вкладкой включая, например, 4 столбца)

Дважды щелкните по одному из полей, чтобы открыть диалоговое окно **Go To Line**. Здесь Вы можете ввести различную позицию, где установить курсор.

Состояние режима редактирования обозначено следующими аббревиатурами:

Аббревиатура	Описание
INS	Режим вставки (Insert mode)
OVR	Режим перезаписи (Overwrite mode)

Дважды щелкните по этому полю, чтобы переключить уставку.

Обозначаются следующие состояния программы:

Текст	Описание
Program loaded	Программа загружена в устройство.
Program unchanged	Программа в устройстве соответствует той, что в системе программирования.
Program modified (Online Change)	Программа в устройстве отличается от имеющейся в системе программирования, требуется онлайновое изменение.
Program modified (Full download)	Программа в устройстве отличается от имеющейся в системе программирования, требуется полная загрузка.

Информация в режиме онлайн

Статус приложения в устройстве:

Текст	Цвет фона	Описание
RUN	Зеленый	Выполнение программы.
STOP	Красный	Программа остановилась.
HALT ON BP	Красный	Программа остановилась на контрольной точке.
Следующее поле состояния доступно, только если контроллер, в зависимости от уставки в описании устройства, поддерживает независимый от цикла мониторинг.		
IN CYCLE	Белый	Указывает, что значения наблюдаемых выражений читаются в пределах одного цикла
OUT OF CYCLE	Красный	Указывает, что выборка значений контролируемых переменных не может быть выполнена в одном цикле.

Окна наблюдения и онлайн виды редакторов

Окна наблюдения и онлайн виды редакторов показывают вид мониторинга POU или определяемого пользователем списка наблюдаемых выражений.

Окна, виды и окна редакторов

В Logic Builder есть 2 различных типа окон:

- Некоторые могут быть прикреплены к любому полю окна SoMachine или могут быть расположены на экране как расстыкованные окна независимо от окна SoMachine. Дополнительно они могут быть скрыты, будучи представленными как вкладка в рамке окна SoMachine (обратитесь к главе Настройка интерфейса пользователя (*см. стр. 45*)). Эти окна выводят на экран информацию, которая не зависит от единственного объекта проекта (например, Messages view или Devices tree). Вы можете получить доступ к ним с помощью меню View (см. SoMachine, Menu Commands, Online Help). Большинство видов включает не конфигурируемую панель инструментов с кнопками для сортировки, просмотра, поиска в окне.
- Другие окна открыты, когда Вы просматриваете или редактируете определенный объект проекта в соответствующем редакторе. Они отображаются в снабженном вкладками окне редактора. Вы не можете скрыть или расстыковать их от окна SoMachine. Вы можете получить доступ к ним с помощью меню Window.

Переключение окон

SoMachine позволяет Вам переключаться между открытыми видами и редакторами. Чтобы переключиться между открытыми видами и редакторами, нажмите клавиши CTRL и TAB одновременно. Открывается окно, в котором перечислены виды и редакторы, которые в настоящее время открыты. Пока клавиша CTRL нажата, окно остается открытым. Используйте клавишу TAB или Клавиши со стрелками одновременно, чтобы выбрать определенный вид или редактор.

Навигаторы с вкладками

Обзор

Навигаторы с вкладками - стандартные компоненты экрана Logic Builder. По умолчанию доступны следующие навигаторы:

- Devices tree (Дерево устройств): Оно позволяет Вам управлять устройствами, на которых работает приложение.
- Applications tree (Дерево приложений): Оно позволяет Вам в одном виде управлять как специфичными для проекта, так и глобальной POU (единица организации программы) и задачами.
- Tools tree (Дерево инструментов): Оно позволяет Вам в одном виде управлять как специфичными для проекта, так и глобальными библиотеками или другими элементами.

Вы можете получить доступ к видам с помощью меню **View**.

Добавление элементов в навигаторы

Корневой узел навигатора представляет программируемое устройство. Вы можете вставить дальнейшие элементы ниже этого корневого узла.

Чтобы добавить элементы к узлу **Навигатора**, просто выберите устройство или объект в аппаратных средствах или каталоге программного обеспечения на правой стороне экрана Logic Builder и перетащите его к **Навигатору** (например, **Devices tree**). Узел или узлы, куда выбранное устройство или объект подходят, автоматически расширяются и отображаются полужирным выделением. Другие узлы, где выбранное устройство или объект не могут быть вставлены, отображаются серым цветом (grayed). Бросьте устройство или объект на подходящий узел, и оно вставится автоматически. Если для устройства или объекта требуются какие-либо последующие элементы, такие как коммуникационные менеджеры, они вставляются автоматически.

Альтернативно, Вы можете выбрать узел в дереве. Если возможно добавить объект к выбранному устройству или объекту, отображается кнопка зеленого плюса. Щелкните по этой кнопке плюса, чтобы открыть меню, дающее элементы, доступные для вставки.

Также можно добавить объект или устройство, щелкая правой кнопкой по узлу в **Навигаторе** и выполняя команду **Add Object** или **Add Device**. Тип устройства, который может быть вставлен, зависит от выбранного в данное время объекта в **Навигаторе**. Например, модули для ведомого устройства (слейва) PROFIBUS DP не могут быть вставлены без вставки ранее надлежащего ведомого коммуникационного устройства. Обратите внимание на то, что только устройства, правильно установленные в локальной системе и соответствующие текущей позиции в дереве, доступны для вставки.

Изменение местоположения объектов

Для изменения местоположения объектов используйте стандартные команды буфера обмена (**Cut (вырезать), Copy (копировать), Paste (вставить), Delete (удалить)**) из меню **Edit**. Альтернативно, Вы можете перетащить выбранный объект мышью, в то время когда левая кнопка мыши (плюс клавиша CTRL для копирования) нажата. Когда Вы добавляете устройства, используя функцию копирования и вставки, новое устройство получает то же имя с постепенно увеличивающимся номером.

Обновление версии устройства

Устройство, которое уже вставлено в **Навигаторы**, может быть обновлено до другой версии или преобразовано в другое устройство.

Обратитесь к описанию отдельных команд:

- Команда **Update Device** (обновления устройства) *(см. стр.71)*
- Команда **Convert Device** (конвертирования устройства) *(см. стр. 73)*

Описание дерева устройств

Каждый объект устройства в дереве **Devices tree** представляет определенный (целевой) аппаратный объект.

Примеры: контроллер, узел сети (fieldbus), шинный соединитель, привод, модуль Вх/Вых.

Устройства и подустройства управляются в дереве **Devices tree**. Другие объекты, которые необходимы для работы приложения на контроллере, сгруппированы в других **Навигаторах**.

- Корневой узел дерева - символьная запись узла: <projectname>
- Конфигурация контроллера определена топологическим расположением устройств в дереве Devices tree. Конфигурация определенного устройства или параметров задачи выполняется в соответствующих диалоговых окнах редактора. Также обратитесь к главе «Конфигурация задачи» *(см. стр. 220)*. Таким образом аппаратная структура отображена и представлена в **Devices tree** соответствующим расположением объектов устройств, позволяя Вам установить сложную неоднородную систему сетевых контроллеров и лежащих в её основе сетей (полевых шин).
- Чтобы добавить устройства, сконфигурированные с DTMs (Device Type Managers) к Вашему проекту, активируйте флажок **Use DTM Connection** в нижней части **Devices tree**. Это даёт эффект добавления узла **FdtConnections** ниже корневого узла дерева. Ниже узла **FdtConnections** автоматически вставляется узел менеджера по коммуникации. Вы можете добавить подходящее DTM устройство к этому узлу. Для получения дополнительной информации обратитесь к руководству пользователя SoMachine Device Type Manager (DTM) *(см. SoMachine, Device Type Manager (DTM), User Guide)*.
- Рассмотрите рекомендации по *Добавлению элементов к Навигаторам* в этой главе.

Пример **Devices tree**:

:

1. Root node (Корневой узел)
2. Programmable device (with applications) (Программируемое устройство (с приложениями))
3. Symbolic device name (Символьное имя устройства)
4. Device name defined in device description file (Имя устройства, определённое в файле описания устройства)

- Каждая запись в дереве **Devices tree** показывает символ, символьное имя (редактируемое), и тип устройства (= имя устройства в соответствии с описанием устройства).
- Устройство программируемо или конфигурируемо. Тип устройства определяет возможную позицию в дереве и также какие дальнейшие ресурсы могут быть вставлены ниже устройства.
- В одном проекте Вы можете сконфигурировать одно или несколько программируемых устройств - независимо от производителя или типа (мульти-ресурс, мульти-устройство, организация сети).
- Сконфигурируйте устройство в части коммуникации, параметров, распределения Вх/Вых в диалоговом окне устройства (редактор устройства). Чтобы открыть редактор устройства, дважды щелкните по узлу устройства в **Devices tree** (обратитесь к описанию редактора устройства *(см. стр. 95)*).
- В режиме онлайн статус устройства показывается иконкой, предшествующей расположению устройства:
  - Контроллер подсоединён, приложение работает, устройство в рабочем состоянии, происходит обмен данными. Опция **Update IO while in stop** (обновление Вх/Вых в режиме **stop**) в виде **PLC settings** редактора устройств *(см. стр. 121)* может быть разрешена или запрещена.
  - Контроллер подсоединён и остановлен (STOP). Опция **Update IO while in stop** (обновление Вх/Вых в режиме **stop**) в виде **PLC settings** редактора устройств *(см. стр. 121)* запрещена.

- Контроллер подсоединён, активное приложение работает, диагностическая информация доступна.
- Устройство не обменивается данными, обнаружена ошибка шины, не сконфигурировано или режим симуляции (обратитесь к описанию команды **Simulation**).
- Устройство работает в демо-режиме 30 минут. После этого времени истекает демонстрационный режим, и сеть прекращает обмениваться данными.
- Устройство сконфигурировано, но не полностью работоспособно. Данные не передаются. Например, устройства CANopen находятся в режиме запуска и предпускового состояния.
- Режим Redundancy активен: ведущее устройство (мастер) сети в настоящее время не посылает данные, потому что другое ведущее устройство находится в активном режиме.
- Описание устройства не было найдено в репозитарии устройств. Для получения дополнительной информации по установке и удалении устройств в диалоговом окне **Device Repository**, обратитесь к описанию **Репозитария Устройств** *(см. SoMachine, Menu Commands, Online Help)*.

- Имена всех подключенных в настоящее время устройств и приложений выведены на экран затенёнными зеленым (shaded) цветом.
- Имена устройств, работающих в режиме симулирования (обращайтесь к описанию команды **Simulation**), выведены на экран курсивом.
- Дополнительная диагностическая информация предоставлена в виде **Status** редактора устройства *(см. стр. 134)*.

Вы можете также запустить активное приложение на устройстве симуляции, которое по умолчанию автоматически доступно в системе программирования. Поэтому никакое реальное целевое устройство не требуется, чтобы протестировать онлайновое поведение приложения (по крайней мере, то, что не полагается на аппаратные ресурсы для своего выполнения). Когда Вы переключаетесь в режим симуляции *(см. SoMachine, Menu Commands, Online Help)*, запись в **Devices tree** показывается курсивом, и Вы можете войти (log into) в приложение.

Вы можете также соединиться с контроллером в онлайновом режиме конфигурации (обращайтесь к главе *«Режим онлайн конфигурации» (см. SoMachine, Menu Commands, Online Help)*) без потребности сначала загрузить реальное приложение в контроллер. Это полезно для начального запуска системы Вх/Вых, потому что Вы можете получить доступ и протестировать Вх/Вых в конфигурации контроллера, прежде чем Вы создадите и загрузите реальную прикладную программу.

Для получения информации о преобразовании ссылок устройств, при открытии проектов, обратитесь к *SoMachine Compatibility and Migration User Guide*.

Расположение и конфигурирование объектов в дереве устройств

Добавление устройств / объектов:
Чтобы добавить устройства или объекты к дереву **Devices tree**, просто выберите устройство или объект в аппаратном каталоге на правой стороне экрана Logic Builder и перетащите его к **Devices tree**. Узел или узлы, куда выбранное устройство или объект подходит, раскрыты и показаны полужирным шрифтом. Другие узлы, куда выбранное устройство или объект не могут быть вставлены, показаны серым шрифтом (grayed). Освободите устройство или объект на подходящем узле, и он вставится автоматически.

Альтернативно, Вы можете выбрать узел в дереве. Если можно добавить объект к выбранному устройству или объекту, отображается кнопка зеленого плюса. Щелкните по этой кнопке плюса, чтобы открыть меню, дающее элементы, доступные для вставки.

Также можно добавить объект или устройство, щелкая правой кнопкой по узлу в **Devices tree** и выполняя команду **Add Object** или **Add Device**. Тип устройства, который может быть вставлен, зависит от выбранного в данное время объекта в **Devices tree**. Например, модули для ведомого устройства (слейва) PROFIBUS DP не могут быть вставлены без вставки ранее надлежащего ведомого коммуникационного устройства. Никакие приложения не могут быть вставлены ниже непрограммируемых устройств.

Обратите внимание на то, что только устройства, правильно установленные в локальной системе и в соответствии с текущей позицией в дереве, доступны для вставки.

Изменение местоположения объектов:
Для изменения местоположения объектов используйте стандартные команды буфера обмена (**Cut, Copy, Paste, Delete**) из меню **Edit**. Альтернативно, Вы можете перетащить выбранный объект мышью, в то время когда левая кнопка мыши (плюс клавиша CTRL для копирования) нажата. Рассмотрим команду **Paste**: В случае, когда вставляемый объект может быть помещён ниже или выше текущей выбранной позиции, открывается диалоговое окно выбора позиции **Select Paste Position.** Это позволяет определить позицию вставки. Когда Вы добавляете устройства, используя функцию копирования и вставки, новое устройство получает то же имя с постепенно увеличивающимся номером.

Обновление версии устройства:
Устройство, которое уже вставлено в **Devices tree**, может быть заменено другой версией того же типа устройства или устройством другого типа (обновление устройства). При этом дерево конфигурации, расположенное ниже соответствующего устройства, сохраняется как можно дольше.

Добавление устройств к корневому узлу:
Только устройства могут быть расположены на уровне непосредственно ниже корневого узла <projectname>. Если Вы выбираете другой тип объекта из диалогового окна **Add Object**, такой как объект **Text list**, он добавляется к узлу **Global** дерева **Applications tree**.

Подузлы (Subnodes):
Устройство вставлено как узел в дереве. Если они определены в файле описания устройства, под-узлы вставляются автоматически. Подузел может быть снова программируемым устройством.

Вставка устройств ниже объекта устройства:

Вы можете вставить дальнейшие устройства ниже объекта устройства. Если они установлены в локальной системе и таким образом доступны в аппаратном каталоге или в диалоговом окне **Add Object** или **Add Device**. Объекты устройства отсортированы в дереве сверху донизу: На определенном древовидном уровне сначала располагаются программируемые устройства, сопровождаемые последующими устройствами – каждые сортированы в алфавитном порядке.

Описание дерева приложений

Объекты **Application**, конфигурация задач и объекты задач управляются в **Applications tree**.

Объекты, нужные для программирования устройства (приложения, текстовые списки, и т.д.), управляются в **Applications tree**. Устройства, которые не программируемы (только конфигурация) не могут быть назначены как программируемые объекты. Вы можете отредактировать значения параметров устройства в диалоговом окне параметров редактора устройства.

Программируемые объекты, такие как конкретная POU или списки глобальных переменных, могут управляться 2 различными способами в **Applications tree**, в зависимости от их объявления:

- Когда они объявлены как под-узел узла **Global**, к этим объектам могут получить доступ все устройства.
- Когда они объявлены как под-узел узла **Applications**, к этим объектам могут получить доступ только соответствующие устройства, объявленные в этом узле Applications.

Вы можете вставить объект **Application** только в дерево **Applications tree**.

Ниже каждого приложения Вы можете вставить дополнительные объекты программирования, такие как **DUT**, **GVL** или объекты визуализации. Вставьте конфигурацию задач ниже приложения. В этой конфигурации задач должны быть определены соответствующие вызовы программ (экземпляры POU из узла **Global** дерева **Applications tree** или специфические для устройства POU). Предполагается, что приложение определено в виде **I/O Mapping** соответствующего редактора устройства *(см. стр. 136)*.

Описание дерева инструментов

Библиотеки управляются в дереве **Tools tree**. Только конфигурируемые устройства не могут быть назначены как объекты программирования. Вы можете отредактировать значения параметров устройства в диалоговом окне параметров редактора устройства.

Программируемые объекты, такие как менеджер библиотек **Library Manager**, могут управляться 2 различными способами в дереве **Tools tree**, в зависимости от их объявления:

- Когда они объявлены как под-узел узла **Global**; тогда к этим объектам могут получить доступ все устройства.
- Когда они объявлены как под-узел узла **Applications**; тогда к этим объектам могут получить доступ только соответствующие устройства, объявленные в этом узле **Applications**.

Вид каталога, снабженного вкладками

Обзор

Hardware Catalog (Каталог аппаратных средств) с вкладками - стандартный компонент экрана Logic Builder. Он содержит следующие вкладки:

- **Controller** (Контроллер): Содержит контроллеры **Logic** (логический), **HMI** (человеко-машинного интерфейса), **Drive** (привода) и **Motion** (движения), которые могут быть вставлены в Ваш проект SoMachine.
- **Devices & Modules** (Устройства & Модули): Содержит **I/O Modules** (модули Вх/Вых), и устройства **Communication** (коммуникации), **Motor Control** (управления приводом), **Safety** (безопасности) и **Energy Management** (управления энергопотреблением), которые могут быть вставлены в Ваш проект SoMachine. Она также позволяет Вам вставлять устройства при помощи шаблона устройства.
- **HMI & iPC**: Содержит HMI и iPC устройства, которые могут быть вставлены в Ваш проект SoMachine.
- **Diverse**: Содержит устройства, сторонних производителей, которые могут быть вставлены в Ваш проект SoMachine.

Содержание отдельных вкладок зависит от проекта. Если контроллеры, интегрированные в проект SoMachine, не поддерживают, например, CANopen, то устройства CANopen не отображаются в каталогах.

Вы можете расширить этот вид вкладками каталога программного обеспечения **Software Catalog** (**Variables** - переменные, **Assets** - активы, **Macros** - макросы, **ToolBox** – панель инструментов, **Libraries** - библиотеки) с помощью меню **View** → **Software Catalog**.

Кнопки **Hardware Catalog** и **Software Catalog** на панели инструментов позволяют Вам отображать или скрывать виды каталогов.
Вы можете добавлять элементы из каталогов в проект простым перетаскиванием в нужное место (drag and drop), как описано в главе «*Добавление устройств способом Drag and Drop* (*см. стр. 60*).

Поиск в каталогах

Каждая вкладка вида каталога содержит поле поиска. Подсписки вкладки ищутся по строке, которую Вы вводите в поле поиска. В открытых подсписках найденные записи отмечаются желтым цветом. Любые другие элементы списка, которые не соответствуют строке поиска, скрыты. Количество элементов, найденных в закрытых подсписках, отображено полужирным шрифтом в строке заголовка каждого подсписка.

По умолчанию поиск выполняется по именам элементов в списках. Но SoMachine также поддерживает механизм тегирования. Это позволяет Вам назначать строки поиска по Вашему выбору любому элементу, включенному в вид **Catalog**.

Список избранного

Каждая вкладка вида каталога содержит список **Favorites** (избранного). Чтобы обеспечить быстрый доступ, Вы можете добавить часто используемые элементы к этому списку с помощью «drag and drop».

Добавление устройств из шаблонов устройств на вкладке Devices & Modules

Вкладка **Devices & Modules** содержит опцию **Device Template** в нижней части. Активируйте эту опцию для отображения доступных шаблонов полевых устройств в списках вкладки **Devices & Modules**. Добавьте их к дереву, как описано в главе «*Добавление устройств из шаблона»* *(см. стр. 746)*.

Настройка интерфейса пользователя

Обзор

Вид интерфейса пользователя, с точки зрения расположения и конфигурации определенных компонентов, зависит от следующего:

- Стандартные предварительные настройки для меню, функций клавиатуры и панелей инструментов. Вы можете перезаписать настройки SoMachine по умолчанию через диалоговое окно **Customize** *(см. SoMachine, Menu Commands, Online Help)* (по умолчанию доступно в меню **Tools**). Текущие настройки сохранены в локальной системе. Функция сброса доступна для восстановления значений по умолчанию в любое время.
- Свойства редактора, как определено в соответствующем диалоговом окне **Tools** → **Options** *(см. SoMachine, Menu Commands, Online Help)*. Вы можете также перезаписать эти параметры настройки. Текущая конфигурация сохранена в локальной системе.
- Последовательность, в которой Вы располагаете виды или окна редактора в проекте. Текущие позиции сохраняются вместе с проектом (см. ниже).
- Выбранная перспектива. По умолчанию выбрана перспектива **Logic Configuration**. Для дополнительной информации обратитесь к параграфу *Перспективы (Perspectives)* в этой главе *(см. стр. 47)*.

Расположение строк меню и панелей инструментов

Строка меню расположена наверху интерфейса пользователя между панелью заголовка окна и окнами видов. Вы можете расположить панель инструментов в той же области, как строку меню (фиксировано) или как независимое окно где угодно на экране.

В окнах видов, таких как дерево **Devices tree**, доступна специальная панель инструментов. Это обеспечивает элементы для сортировки, просмотра и поиска в окне. Вы не можете конфигурировать эту панель инструментов.

Расположение окон и видов

Закрытие вида или окна редактора: Нажмите кнопку крестика в правом верхнем углу.

Открытие закрытого вида: По умолчанию Вы можете вновь открыть виды стандартных компонентов с помощью меню **View**. Чтобы открыть окно редактора, выполните команду **Project** → **Edit object** или дважды щелкните по соответствующей записи в **Devices tree**, **Applications tree** или в **Tools tree**.

Изменение размеров вида или окна в рамочном (frame) окне: Переместите линии разделителя между соседними видами. Вы можете изменить размеры независимых окон видов на рабочем столе, переместив границы окна.

Перемещение вида к другой позиции на рабочем столе или в рамочном окне: Щелкните по строке заголовка или, в случае снабженных вкладками видов альтернативно по вкладке вида, удерживайте кнопку мыши в нажатом состоянии и переместите вид в желаемое место. Символы стрелок будут показывать каждую возможную целевую позицию. Удерживайте кнопку мыши в нажатом состоянии и выберите желаемую позицию, переместив курсор на соответствующий символ стрелки. Целевая позиция обозначена затененной синим цветом областью.

Символы стрелок, указывающие новую позицию

Символ стрелки	Описание
	Вид помещается выше.
	Вид помещается ниже.
	Вид помещается направо.
	Вид помещается налево.
	Вид помещается сюда: вид, который в настоящее время помещен в эту позицию, и новый вид располагаются как значки.

Пример навигации символами стрелок

Когда Вы отпускаете кнопку мыши, вид помещается в новую позицию.

Виды с кнопкой **Auto Hide** могут быть помещены как независимые окна (плавающие) куда угодно на экране перемещением их, а не перетаскиванием их на один из символов стрелок. В этом случае вид теряет кнопку **Auto Hide**. В качестве альтернативы выполните команды **Dock** и **Float** из меню **Window**.

Скрытие видов: Вы можете скрыть виды с кнопками **Auto Hide** на границе окна SoMachine. Нажмите кнопку **Auto Hide** вниз в правом верхнем углу вида. Вид будет отображён как вкладка на ближайшей границе рамочного окна. Содержание вида видимо, только пока курсор перемещен в эту вкладку. Вкладка выводит на экран значок и имя вида. Это состояние вида обозначено прикрепляющей кнопкой, измененной на **Auto Hide** влево.

Вывод видов на экран: Чтобы вывести на экран вид, нажмите левую кнопку Auto Hide.

Альтернативный способ скрыть и вывести на экран вид обеспечен командой **Auto Hide**, которая по умолчанию доступна в меню **Window**.

Невозможно изменить местоположение информации и строки состояния на нижней границе интерфейса пользователя *(см. стр. 33)*.

Перспективы

Перспектива используется, чтобы сохранить расположение видов SoMachine. Она хранит информацию о том, открыты ли виды **Messages и Watch** и в каких позициях располагаются окна видов (прикрепленные или независимые окна).

По умолчанию SoMachine обеспечивает 4 перспективы для следующих вариантов использования в меню **Window** → **Switch Perspective** или в таблице перспектив на панели инструментов.

Имя перспективы	Вариант использования	Навигаторы (на левой стороне)	Каталог видов (на правой стороне)	Виды внизу экрана
Device Configuration	Для добавления / конфигурирования устройств.	• **Devices tree** • **Applications tree** • **Tools tree**	Аппаратный каталог • **Controller** • **Devices & Modules** • **HMI & iPC** • **Diverse**	**Messages** (в режиме **Auto Hide**)
Logic Configuration	Для добавления / создание логики.	• **Devices tree** • **Applications tree** • **Tools tree**	Каталог программного обеспечения **Variables** **Assets** **Macros** **ToolBox** **Libraries**	**Messages** (в режиме **Auto Hide**)
CODESYS Classic	Стандартные виды CoDeSys.	• **Devices** • **POUs**	Аппаратный каталог • **Controller** • **Devices & Modules** • **HMI & iPC** • **Diverse**	**Messages** (в режиме **Auto Hide**)

Имя перспективы	Вариант использования	Навигаторы (на левой стороне)	Каталог видов (на правой стороне)	Виды внизу экрана
Online	Для режима онлайн.	• **Devices tree** • **Applications tree** • **Tools tree**	Аппаратный каталог • **Controller** • **Devices & Modules** • **HMI & iPC** • **Diverse**	**Messages** (в режиме **Auto Hide**) **Watch 1**

Перспектива **Online** выбирается автоматически, когда приложение переключается в режим онлайн.

Создание собственной перспективы:

В дополнение к этим стандартным перспективам Вы можете создать свое собственное расположение видов и сохранить его в различных перспективах согласно Вашим индивидуальным требованиям.

Чтобы создать Вашу собственную перспективу, продолжите следующим образом:

Шаг	Действие
1	Измените размеры, откройте или закройте виды согласно Вашим индивидуальным требованиям.
2	Выполните команду **Save current view layout as perspective** из меню **Window**, чтобы сохранить Ваши модификации в новой перспективе.
3	В диалоговом окне **Save current view layout as perspective** введите **Name (**Имя) для своей перспективы. **Результат:** текущее расположение видов сохранено. Новая перспектива доступна в меню **Window** → **Switch Perspective** и в таблице перспектив на панели инструментов.

Сброс перспективы к ее начальному состоянию:

Чтобы сбросить измененную перспективу к ее начальному состоянию, выполните команду **Reset current Perspective** из меню **Window**.

Импорт / экспорт перспектив:
Чтобы быть в состоянии обмениваться перспективами между различными установками SoMachine или между различными пользователями, диалоговое окно **Tools** → **Options** → **Perspectives** *(см. SoMachine, Menu Commands, Online Help)* позволяет Вам экспортировать перспективы в XML-файл и импортировать уже доступные XML-файлы перспектив.

Изменение масштаба (Zoom)

Каждое окно редактора обеспечивает функцию изменения масштаба. Нажмите кнопку изменения масштаба в правом нижнем углу окна, чтобы открыть список. Это позволяет Вам выбирать один из уровней 25, 50, 100, 150, 200 и 400 процентов или вводить фактор изменения масштаба по Вашему выбору. Распечатка всегда относится к 100%-му виду.

Настройка интерфейса пользователя возможна в режиме офлайн и в режиме онлайн.

Интерфейс пользователя в режиме онлайн

Обзор

Как только Вы входите с проектом в систему, объекты, которые уже были открыты в режиме офлайн, автоматически просматриваются в режиме онлайн. Перспектива автоматически переключается на перспективу **Online** *(см. стр. 47)*, что означает, что вид **Watch** открывается по умолчанию.

Чтобы открыть в режиме онлайн объект, который ещё не открыт, дважды щелкните по узлу в дереве **Applications** или выполните команду **Project** → **Edit Object**. Объект будет открыт в режиме онлайн.

Если есть несколько экземпляров выбранного объекта (таких как функциональные блоки), содержащихся в проекте, откроется диалоговое окно под названием **Select Online State <object name>**. Оно позволяет Вам выбирать, должны ли экземпляр или основная реализация объекта быть отображены и должен ли объект быть отображён в режиме онлайн или режиме офлайн.

Диалоговое окно **Select Online State**

Поле **Device/Application** содержит устройство и приложение, с которыми связан соответствующий объект.

Чтобы открыть онлайн вид объекта, активируйте опцию **Online mode** и нажмите **OK**. Чтобы видеть офлайн вид, активируйте опцию **Offline mode**.

Если объект - функциональный блок, поле **Function block instance** содержит список экземпляров, в настоящее время используемых в приложении.

В этом случае доступны опции:

- Или выберите один из экземпляров и активируйте режим **Online** или **Offline mode**.
- Или выберите опцию **Implementation**, которая - независимо от выбранного экземпляра - откроет основной вид реализации функционального блока. У опции **Implementation** нет влияния на неустановленные (non-instantiated) объекты.

Для получения дополнительной информации об онлайновых видах определенных редакторов обратитесь к соответствующим описаниям редакторов.

Строка состояния (*см. стр. 33*) предоставляет информацию о текущем статусе приложения.

Меню и команды

Обзор

Следующий рисунок показывают строку меню по умолчанию:

Некоторые команды не видимы в виде по умолчанию. Чтобы добавить команду к меню, вставьте её в меню по Вашему выбору при помощи диалогового окна **Tools** → **Customize** *(см. SoMachine, Menu Commands, Online Help)*.

Определенные команды, для определенного редактора, например, обычно доступны в соответствующем меню. Эти команды видимы только тогда, когда редактор открыт. Например: когда Вы редактируете объект в редакторе SFC, меню SFC добавлено к строке меню.

Чтобы реорганизовать структуры меню, используйте диалоговое окно **Tools** → **Customize**.

Несколько команд меню **File** не доступны, потому что эти задачи выполняются в SoMachine Central. Для получения дополнительной информации обращайтесь к SoMachine Central User Guide *(см. SoMachine Central, User Guide)*.

Для описания меню и команд обратитесь к отдельным онлайн-справкам меню команд SoMachine *(см. SoMachine, Menu Commands, Online Help)*.

Глава 3

Фундаментальные понятия

Введение и фундаментальные понятия

Обзор

SoMachine – не зависящая от устройств система программирования контроллеров.

Соответствующая стандарту IEC 61131-3, она поддерживает все стандартные языки программирования. Кроме того, она позволяет включать C-подпрограммы. Это позволяет Вам программировать многочисленные устройства управления в одном проекте.

Для получения дополнительной информации см. главу **Generate runtime system files** *(см. SoMachine, Menu Commands, Online Help)*.

Объектная ориентация

Объектно-ориентированный подход не только отражен доступностью надлежащих программных элементов и функций, но также и в структуре, и обработке версии SoMachine, и в организации работ по проекту. Использование проекта SoMachine для нескольких устройств возможно на основе совместно используемых, установленных программных элементов. Клонирование приложений возможно также как смешивание конфигурируемых и программируемых устройств управления в одном проекте.

Обработка версии

Одновременная установка нескольких версий компонентов SoMachine и работа с желаемой комбинацией версий возможна. Это также относится к использованию специфичных для устройств различных версий компилятора. Отдельные функции могут быть добавлены, без необходимости обновления целой версии. Для получения дополнительной информации обратитесь к *SoMachine Compatibility and Migration User Guide* (Руководству пользователя по совместимости и миграции SoMachine).

Организация работ по проекту

Организация работ по проекту также определена объектно-ориентированным способом. Проект SoMachine содержит программу контроллера, состоящую из различных объектов программирования, и он содержит определения ресурсов, которые необходимы, чтобы выполнить экземпляры программы (приложения) в определенных целевых системах (устройства, контроллеры).

Таким образом, в проекте есть два основных типа объектов:

Тип объекта	Описание
Объекты программирования (POU) (*см. стр.149*):	Это программы, функции, функциональные блоки, методы, интерфейсы, действия, типы данных, определения и т.д. Объекты программирования, которые можно инстанцировать во всем проекте, т.е. для всех приложений, определенных в проекте, должны управляться в глобальном (**Global)** узле дерева **Applications tree**. Инстанцирование делается вызовом программы POU назначенной приложению задачей. Объекты программирования, которыми управляют только в дереве **Applications tree**, т.е. которые непосредственно присвоены приложению, не может инстанцировать только другое приложение, вставленное ниже.
Объекты ресурсов (**Devices tree**)	Объекты устройств управляются только в дереве **Devices tree**. Когда Вы вставляете объекты в **Devices tree**, рассмотрите рекомендации, описанные в разделе *Навигаторы добавления элементов (см. стр. 41)*.

Генерация кода

Генерация кода интегрированными компиляторами с последующим использованием получающегося машинного кода обеспечивает короткие времена выполнения.

Передача данных в устройство контроллера

Передача данных между SoMachine и устройством проводится через шлюз (компонент) и систему реального времени. Доступна полная онлайн функциональность для управления программой в устройстве.

Поддерживаемые языки программирования

Языки программирования, упомянутые в стандарте IEC 61131, поддерживаются посредством специально адаптированных редакторов:

- Редактор FBD/LD/IL (*см. стр. 249*) для схем функциональных блоков (FBD), схем лестничной логики (LD), и списка инструкций (IL).
- Редактор SFC *(см. стр. 317)* для диаграмм последовательных функций.
- Редактор ST *(см. стр. 347)* для структурированного текста.

Кроме того, SoMachine предоставляет редактор для программирования в CFC, который не является частью стандарта IEC:
Редактор CFC *(см. стр. 297)* для непрерывной диаграммы функций

CFC является расширением стандартных языков программирования IEC.

Часть II

Конфигурация

Содержание этой части

Эта часть содержит следующие главы:

Глава	Наименование главы	Страница
4	Установка устройств	57
5	Управление устройствами	59
6	Общие диалоговые окна редактора устройств	93

Глава 4

Установка устройств

Интеграция Sercos устройств от сторонних поставщиков

Введение

Через диалоговое окно **Device Repository** (см. *SoMachine, Menu Commands, Online Help*), Вы можете интегрировать устройства Sercos с универсальными профилями ввода-вывода в Вашу систему программирования.

Чтобы установить это Sercos устройство, Вам нужен SDDML (Sercos Device Description Markup Language) файл (описание устройства для Sercos устройств) обеспеченный поставщиком устройства. Файл SDDML - файл описания устройства для Sercos устройств.

Есть два типа Sercos устройств с универсальными доступными профилями ввода-вывода:

- Блочные устройства ввода-вывода
  Блочное устройство ввода-вывода - заранее смонтированный блок, который состоит из шинного интерфейса и модуля ввода-вывода.
- Модульные устройства ввода-вывода
  Модульные устройства ввода-вывода - модули ввода-вывода, которые могут быть соединены с шинным интерфейсом.

Интегрирование в SoMachine

Чтобы интегрировать устройства Sercos от сторонних поставщиков в Вашу систему программирования выполните следующее:

Шаг	Действие
1	Выберите **Tools** → **Device Repository...** из строки меню. **Результат:** открывается диалоговое окно **Device Repository**.
2	Нажмите кнопку **Install...** в диалоговом окне **Device Repository**. **Результат**: открывается диалоговое окно **Install Device Description**.
3	Выберите тип файла **SERCOS III I/O device descriptions (*.xml)** и просмотрите Вашу файловую систему для открытия файла SDDML.
4	Выберите файл SDDML и нажмите **Open**. **Результат**: файл SDDML преобразован и импортирован в формат файла, совместимый с SoMachine.

ПРИМЕЧАНИЕ: Если выбранный файл SDDML не совместим или если устройство Sercos стороннего поставщика не использует тип совместимый с FSP (Function Specific Profile), то в виде **Messages** будет обозначено соответствующее диагностическое сообщение (см. *SoMachine, Menu Commands, Online Help*).

Проверка интеграции

Чтобы проверить, было ли устройство Sercos с универсальным профилем ввода-вывода интегрировано в Вашу систему программирования, выполните следующее:

Шаг	Действие
1	Выберите **Tools** → **Device Repository...** из строки меню. **Результат**: открывается диалоговое окно **Device Repository**.
2	В древовидной структуре описания установленных устройств (**Installed device descriptions)** разверните узел **Fieldbusses** → **Sercos.**
3	Разверните подузел **Slave,** чтобы проверить, доступны ли в списке шинные интерфейсы Sercos, которые Вы интегрировали.
4	Разверните подузел **Module**, чтобы проверить, доступны ли в списке модули ввода-вывода Sercos, которые Вы интегрировали,

Для получения дополнительной информации обратитесь к описанию диалогового окна **Device Repository** (см. *SoMachine, Menu Commands, Online Help*).

Глава 5

Управление устройствами

Содержание этой главы

Эта глава содержит следующие разделы:

Раздел	Тема	Страница
5.1	Добавление устройств перетаскиванием (Drag and Drop)	60
5.2	Добавление устройств из контекстного меню или кнопкой Plus (плюс)	63
5.3	Обновление устройств	71
5.4	Преобразование устройств	73
5.5	Преобразование проектов	77

Раздел 5.1

Добавление устройств перетаскиванием (Drag and Drop)

Добавление устройств перетаскиванием (Drag and Drop)

Обзор

SoMachine V4.0 и более поздние версии обеспечивают вид каталога с вкладками на правой стороне Logic Builder SoMachine.

Доступны 2 типа вида каталога:

- **Hardware Catalog** (Аппаратный каталог)
- **Software Catalog** (Каталог программного обеспечения)

Чтобы добавить устройство к дереву **Devices tree**, выберите соответствующую запись в **Hardware Catalog**, перетащите его в **Devices tree** и оставьте его в подходящем узле. Оно автоматически добавится к Вашему проекту.

Добавление контроллеров перетаскиванием (Drag and Drop)

Для добавления контроллеров в Ваш проект выполните следующее:

Шаг	Действие
1	Откройте **Hardware Catalog**, нажав кнопку аппаратного каталога на панели инструментов Logic Builder SoMachine, если он уже не открыт.
2	Выберите вкладку **Controller** в **Hardware Catalog (аппаратном каталоге)**. **Результат**: контроллеры, подходящие для Вашего проекта SoMachine, выведены на экран в **Hardware Catalog**.
3	Выберите компонент контроллера на вкладке **Controller**, перетащите его к **Devices tree** и оставьте его в подходящем узле. Вы можете освободить контроллер в любом свободном месте **Devices tree**. **Результат**: контроллер добавлен к **Devices tree** как новый узел с различными подузлами в зависимости от типа контроллера.

Добавление устройств расширения перетаскиванием (Drag and Drop)

Для добавления устройства расширения к контроллеру, выполните следующее:

Шаг	Действие
1	Откройте **Hardware Catalog**, нажав кнопку аппаратного каталога на панели инструментов SoMachine Logic Builder, если он уже не открыт.
2	Выберите вкладку **Devices & Modules** в **Hardware Catalog**. **Результат**: устройства расширения, подходящие для Вашего проекта SoMachine, отображаются в **Hardware Catalog**.
3	Выберите свое устройство расширения, перетащите его к **Devices tree** и освободите его в подходящем подузле контроллера. **ПРИМЕЧАНИЕ**: Подходящие подузлы раскрыты и выделены SoMachine. **Результат**: устройство расширения добавлено к **Devices tree** ниже подузла контроллера.
4	Если устройство расширения требует менеджера по коммуникации, этот узел добавлен автоматически к **Devices tree**. Если для Вашего устройства расширения доступны несколько менеджеров по коммуникации, на экран будет выведено диалоговое окно, которое позволит Вам выбрать подходящего менеджера по коммуникации.

Добавление устройств и модулей перетаскиванием (Drag and Drop)

Чтобы добавить полевое устройство к контроллеру, выполните следующее:

Шаг	Действие
1	Откройте Hardware Catalog, нажав кнопку аппаратного каталога на панели инструментов Logic Builder SoMachine, если он уже не открыт.
2	Выберите вкладку **Devices & Modules** в **Hardware Catalog**. **Результат**: полевые устройства, подходящие для Вашего проекта SoMachine, будут отображены в **Hardware Catalog**.
3	Выберите компонент полевого устройства в виде каталога **Devices & Modules**, перетащите его к **Devices tree** и освободите его в подходящем подузле контроллера. **ПРИМЕЧАНИЕ**: Подходящие подузлы раскрыты и выделены SoMachine. **Результат**: полевое устройство добавлено к дереву **Devices tree** ниже подузла контроллера.
4	Если полевое устройство требует менеджера по коммуникации, этот узел добавлен автоматически к **Devices tree**. Если для Вашего полевого устройства доступны несколько менеджеров по коммуникации, на экран будет выведено диалоговое окно, которое позволит Вам выбрать подходящего менеджера по коммуникации.

Добавление устройств из шаблона устройств перетаскиванием (Drag and Drop)

Для добавления устройства из шаблона устройства выполните следующее:

Шаг	Действие
1	Откройте **Hardware Catalog**, нажав кнопку аппаратного каталога на панели инструментов Logic Builder SoMachine, если он уже не открыт.
2	Выберите вкладку **Devices & Modules** в **Hardware Catalog**.
3	Выберите опцию **Device Template** внизу вкладки **Devices & Modules**. **Результат**: шаблоны устройства, подходящие для Вашего проекта SoMachine, отображаются на вкладке **Devices & Modules**.
4	Добавьте их к **Devices tree**, как описано в главе *Добавление устройств из шаблона* *(см. стр. 746)*.

Добавление устройств из шаблона функций перетаскиванием (Drag and Drop)

Для добавления устройства из шаблона функций выполните следующее:

Шаг	Действие
1	Откройте каталог программного обеспечения, нажав кнопку **Software Catalog** на панели инструментов Logic Builder SoMachine, если он уже не открыт.
2	Выберите вкладку **Macro** в **Software Catalog**. **Результат**: шаблоны функций, доступные в SoMachine, отображаются в **Software Catalog**.
3	Выберите запись шаблона функции в виде **Macro**, перетащите её к дереву **Devices tree** и освободите её в подходящем подузле контроллера. **ПРИМЕЧАНИЕ**: Подходящие подузлы раскрыты и выделены SoMachine. **Результат**: устройство на основе шаблона функции добавлено к **Devices tree**.

Раздел 5.2

Добавление устройств из контекстного меню или кнопкой Plus (плюс)

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Добавление контроллера	64
Добавление устройств расширения	65
Добавление менеджеров по коммуникации	66
Добавление устройств менеджеру по коммуникации	68
Добавление устройств из шаблона	70

Добавление контроллера

Введение

Как альтернатива перетаскивания устройств на дереве **Devices tree**, щелкните по кнопке зеленого плюса, которая отображается в подходящем узле в **Tree**. Альтернативно Вы можете щелкнуть правой кнопкой по узлу **Tree**, чтобы добавить подходящее устройство, используя контекстное меню. Открывается диалоговое окно **Add Device**, которое позволяет Вам определять, будет ли устройство добавлено, вставлено или включено к выбранному узлу *(см. SoMachine, Menu Commands, Online Help)*.

Когда Вы добавляете контроллер к своему проекту, несколько узлов автоматически добавляются к дереву **Devices tree**. Эти подузлы специфичны для контроллера, в зависимости от функций, которые он обеспечивает.

Следующий параграф описывает общую процедуру добавления контроллера. Для получения дополнительной информации об определенном контроллере, обратитесь к руководству по программированию для этого Вашего определенного контроллера.

Добавление контроллера

Чтобы добавить устройство к Вашему проекту SoMachine, выполните следующее:

Шаг	Действие
1	Выберите узел проекта, щелкните по кнопке зеленого плюса узла, или щелкните правой кнопкой по узлу проекта и выберите команду **Add Device...** из контекстного меню. **Результат:** открывается диалоговое окно **Add Device**.
2	В диалоговом окне **Add Device** выберите **Schneider Electric** из поля списка **Vendor**.
3	Выберите контроллер, который Вы хотите вставить в проект.
4	Переименуйте свое устройство, введя имя в текстовое поле **Name**. **ПРИМЕЧАНИЕ**: Выберите имя, которое соответствует стандарту IEC. Не используйте специальные символы, первые цифры или пробелы в имени. Имя не должно превышать длину 32 символов. Если Вы не переименовываете устройство, имя даётся по умолчанию. Содержательное наименование устройства может упростить организацию Вашего проекта.
5	Нажмите кнопку **Add Device**. **Результат:** выбранный контроллер добавлен к проекту и появляется как новый узел в дереве **Devices**. Диалоговое окно **Add Device** остается открытым. Вы можете сделать следующее: ● добавить другой контроллер, возвращаясь к шагу 3. ● или нажать кнопку **Close**, чтобы закрыть диалоговое окно **Add Device**.

Добавление устройств расширения

Доступные устройства расширения

Для получения списка устройств расширения, доступных для различных контроллеров, обратитесь к главе *Поддерживаемые устройства* этого документа.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННАЯ РАБОТА ОБОРУДОВАНИЯ

- Используйте только программное обеспечение, утвержденное Schneider Electric для использования с этим оборудованием.
- Обновляйте свою прикладную программу каждый раз, когда Вы изменяете физическую аппаратную конфигурацию.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

Добавление устройств расширения

Для добавления устройств расширения к Вашему устройству выполните следующее:

Шаг	Действие
1	Выберите узел контроллера и щелкните по кнопке зеленого плюса узла или щелкните правой кнопкой по узлу контроллера и выберите команду **Add Device...** из контекстного меню. **Результат**: открывается диалоговое окно **Add Device**.
2	В диалоговом окне **Add Device** выберите **Schneider Electric** из списка **Vendor (п**оставщиков).
3	Выберите устройство расширения, которое Вы хотите добавить к Вашему контроллеру из приведенного ниже **Device** списка.
4	Переименуйте свое устройство расширения, введя имя в текстовое поле Name. **ПРИМЕЧАНИЕ**: имя не должно содержать пробелы. Если Вы не переименовываете устройство расширения, имя дается по умолчанию. Содержательное наименование устройства расширения может упростить организацию Вашего проекта.
5	Нажмите кнопку **Add Device**. Результат: выбранное устройство расширения добавлено к проекту и выведено на экран в дереве **Devices Tree** как новый подузел Вашего контроллера. Диалоговое окно **Add Device** остается открытым. Вы можете сделать следующее: ● добавить другое устройство расширения, возвращаясь к шагу 3 этого описания. ● или нажать кнопку **Close**.

ПРИМЕЧАНИЕ: Когда Вы добавляете объект TWDNOI10M3 (AS-Interface Master Module), соответствующий менеджер шины **Virtual AS interface bus** будет вставлен автоматически. Для получения дополнительной информации по конфигурации интерфейса AS, обратитесь к главе в Руководстве по программированию логического контроллера Modicon M238 (см. *Modicon M238 Logic Controller, Programming Guide*).

Конфигурация устройства расширения

Для получения большего количества информации о конфигурации обратитесь к *Руководству по программированию* для Вашего устройства расширения.

Добавление менеджеров коммуникаций

Обзор

Существуют 2 типа менеджеров коммуникации:

- менеджеры полевых шин (Fieldbus), которые позволяют конфигурировать устройства этих шин (например, ведомых устройств CANopen slaves или Modbus slaves)
- менеджеры общей коммуникации

Менеджеры коммуникации, доступные в SoMachine, перечислены ниже:

Имя	Тип интер-фейса	Описание
ASCII Manager	Последов. линия	Используется для передачи и/или получения данных от простого устройства.
SoMachine-Network Manager	● Последов. линия (max. 1) ● Ethernet (max. 3)	Используйте его, если Вы хотите соединить XBTGT, XBTGK, XBTGH или SCU HMI через протокол SoMachine, предлагающий прозрачный обмен данными и возможность многочисленной загрузки (загрузка приложений контроллера и HMI через единственное соединение PC–контроллер или PC–HMI). Доступны максимум 4 соединения: 1 – для SoMachine (даже если используется USB–соединение), 3 – для Ethernet.
Modbus IOScanner	Последов. линия	Менеджер протокола Modbus RTU или ASCII используется для определения неявных обменов (сканирование Вх/Вых) с ведомыми устройствами Modbus.
Modbus Manager	Последов. линия	Используется для протокола Modbus RTU или ASCII в ведущем (master) или ведомом (slave) режиме.
CANopen Optimized	CAN	Менеджер CANopen для оптимизированных контроллеров (M238, M241, XBTGC, XBTGT, XBTGK, HMI SCU, ATV IMC)
CANopen Performance	CAN	Менеджер CANopen для производительных контроллеров (M251, M258, LMC058 и LMC078)
CANmotion	CAN	Менеджер CANmotion для контроллеров движения LMC058 и LMC078 только по порту CAN1.
Modbus TCP Slave Device	Ethernet	Менеджер Modbus TCP для контроллеров с портом Ethernet.
EtherNet/IP	Ethernet	Менеджер Ethernet/IP для контроллеров с портом Ethernet (M251, M258, LMC058 и LMC078)

Добавление менеджера по коммуникации

Менеджеры по коммуникации автоматически добавляются с соответствующим устройством.

Чтобы добавить менеджера коммуникации отдельно, продолжите следующим образом:

Шаг	Действие
1	В дереве устройств **Devices Tree** выберите шинный интерфейс (**Serial Line**, **CAN0**, **CAN1**, **Ethernet**) и щелкните по кнопке зеленого плюса узла или щелкните правой кнопкой по узлу шинного интерфейса и выполните команду **Add Device...** из контекстного меню. **Результат**: открывается диалоговое окно **Add Device**.
2	В диалоговом окне **Add Device** выберите **Schneider Electric** из поля списка **Vendor**. **Примечание**: Вы можете сортировать устройства по брендам, щелкнув по полю списка **Vendor**.
3	Выберите **Communication manager** из списка ниже.
4	Переименуйте свое устройство, введя имя в текстовое поле **Name**. **Примечание**: не используйте пробелы в имени. Если Вы не переименовываете устройство, имя даётся по умолчанию. Содержательное наименование устройства может упростить организацию Вашего проекта.
5	Нажмите кнопку **Add Device**.
6	Нажмите кнопку **Close**, чтобы закрыть диалоговое окно **Add Device**.
7	Сконфигурируйте **Communication manager**.

Добавление устройств к менеджеру коммуникации

Обзор

Вы можете добавить полевые устройства к менеджеру коммуникации, выбрав узел менеджера полевого устройства (например, CANopen или Modbus manager) в дереве **Devices Tree** и щелкнув по знаку зеленого плюса. Альтернативно, Вы можете щелкнуть правой кнопкой по узлу менеджера полевого устройства в **Devices Tree** и выполнить команду **Add Device**.

В качестве предпосылки, устройство должно быть доступным в диалоговом окне **Device Repository** *(см. SoMachine, Menu Commands, Online Help).*

Добавление устройств

Шаг	Действие
1	Выберите узел менеджера полевого устройства (CANopen или Modbus manager) в дереве **Devices Tree** и щелкните по знаку зеленого плюса, или щелкните правой кнопкой по узлу менеджера полевого устройства и выберите команду **Add Device**... из контекстного меню. **Результат**: открывается диалоговое окно **Add Device**.
2	В диалоговом окне **Add Device** выберите **Schneider Electric** из поля списка **Vendor**. **Примечание**: Вы можете отсортировать устройства по брендам, щелкнув по полю списка **Vendor**.
3	Выберите устройство по Вашему выбору из списка ниже.
4	Переименуйте свое устройство, введя имя в текстовое поле **Name**. **Примечание**: не используйте пробелы в имени. Если Вы не переименовываете устройство, имя даётся по умолчанию. Содержательное наименование устройства может упростить организацию Вашего проекта.
5	Нажмите кнопку **Add Device**. **Результат**: полевое устройство добавлено к менеджеру полевого устройства. **ПРИМЕЧАНИЕ**: диалоговое окно **Add Device** остается открытым. Вы можете сделать следующее: ● добавить другое устройство, возвращаясь к шагу 2. ● нажать кнопку **Close**.

Доступ к диагностической информации

Для получения диагностической информации устройств CANopen, используйте CAA_CiA405.library.

Доступ к диагностике конфигурации (для передовых пользователей)

Вы можете использовать опции **Abort if error** и **Jump to line if error** во вкладке **Service Data Object** конфигуратора CANopen, чтобы управлять потенциальными несоответствиями конфигурации.

Чтобы оптимизировать производительность ведущего устройства CAN, диагностики CAN являются внешними относительно ведущего устройства CAN в контроллере. Структура диагностики CAN определена в библиотеке CanConfig Extern, доступной в менеджере библиотеки **Library Manager**.

Структура `g_aNetDiagnosis` содержит последнюю диагностическую информацию от ведомых устройств. Структура обновляется каждый раз, когда ведомое устройство сконфигурировано по любой причине.

Эта структура может использоваться в программе, чтобы сделать следующее:

- Мониторинг ответа ведомых устройств, сконфигурированных с помощью сообщений SDO.
- Мониторинг ведущего устройства для любых аварийных сообщений от ведомых устройств, прежде чем разрешить запуск машины / приложения.

Эта структура должна существовать и быть активной в приложении пользователя во время тестирования, отладки и ввода в действие приложения. Когда машина и ее приложение управления будут введены в эксплуатацию и проверены, тогда можно будет отключить этот код от выполнения, чтобы уменьшить трафик в сети CANopen.

Однако, если в течение жизненного цикла приложения и машины или процесса, которым оно управляют, ведомые устройства добавлены или заменены в операционной системе, тогда диагностическая структура должна продолжать оставаться активной в приложении.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННАЯ РАБОТА ОБОРУДОВАНИЯ

- Используйте g_aNetDiagnosis структуру данных в приложении, чтобы контролировать ответы ведомых (slave) устройств CAN на команды конфигурации.
- Проверьте, что приложение не запускает или переводит машину или процесс в рабочее состояние в случае получения аварийных сообщений отбрасывания SDO от любого из ведомых CAN устройств.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования

После добавления библиотеки CanConfig Extern к Вашему приложению используйте определение **Net Diagnostic** в своем приложении, чтобы протестировать на сообщения аварийного прекращения работы SDO от ведомых CAN устройств.

Следующий пример кода иллюстрирует использование структуры данных диагностики CAN:

```
IF g_aNetDiagnosis[CAN_Net_Number].ctSDOErrorCounter = 0 THEN
    (* No error is detected in the configuration*)
ELSE
    (* An error has been detected during configuration. Get the latest error
information.*)
    // node ID of the slave which sent the abort code
    ReadLastErrorNodeID := g_aNetDiagnosis[CAN_Net_Number].usiNodeID;
    // index used in the aborted SDO
    ReadLastErrorIndex := g_aNetDiagnosis[CAN_Net_Number].wIndex;
    // subIndex used in the aborted SDO
    ReadLastErrorSubIndex := g_aNetDiagnosis[CAN_Net_Number].bySubIndex
;
    //SDO abort code
    ReadLastErrorSdoAbortCode := g_aNetDiagnosis [CAN_Net_Number].udiAb
ortCode;
    (* Do not allow the start-
up or other operation of the machine or process *)
END_IF
```

ПРИМЕЧАНИЕ: В этом примере `CAN_Net_Number` был бы 0 для порта CAN0, и, если контроллер так оборудован, 1 для CAN1port.

Добавление устройств из шаблона

Обзор

Также можно добавить новое устройство, используя шаблон устройства. Для описания этой процедуры обратитесь к разделу *Управление шаблонами устройств* (*см. стр. 746*).

Раздел 5.3

Обновление устройств

Обновление устройств

Введение

Функция устройства обновления позволяет Вам заменять устройство, выбранное в **Devices tree**

- другой версией того же устройства или
- другим типом устройства.

Обновление устройств

Чтобы заменить устройство Вашего проекта SoMachine другой версией или другим устройством, продолжите следующим образом:

Шаг	Действие
1	Выберите устройство, которое Вы хотите заменить в **Devices tree** и выполнить команду **Update Device...** из меню **Project**. ИЛИ Щелкните правой кнопкой по устройству, которое Вы хотите заменить в **Devices tree** и выберите команду **Update Device...** из контекстного меню. **Результат**: открывается диалоговое окно **Update Device**. ИЛИ Щелкните правой кнопкой по устройству, которое Вы хотите заменить в **Devices tree** и выбрать команду **Add Device...** из контекстного меню. В диалоговом окне **Add Device** выберите **Action: Update device**. **Результат**: диалоговое окно **Add Device** преобразовано в диалоговое окно **Update Device**.
2	Из **Device:** список выберите устройство, которое должно заменить текущее устройство. Чтобы выбрать определенную версию устройства, выберите опции **Display all versions** (только для экспертов) и/или **Display outdated versions**.
3	Если необходимо отличить устройства, переименуйте свое устройство, введя имя в текстовое поле **Name**. Иначе то же имя будет использоваться для обновленного устройства. Если устройство будет обновлено другим типом устройства, то описание типа устройства (в скобках позади имени устройства) будет автоматически адаптировано. Содержательное наименование устройства может упростить организацию Вашего проекта.
4	Нажмите кнопку **Update Device**. **Результат**: устройство, которое было выбрано в **Devices tree**, заменено новым типом устройства или новой версией. Новый тип устройства или новая версия теперь выведены на экран в выбранном узле в **Devices tree**.

Эффекты после обновления Устройства

Дополнительные устройства, которые расположены в дереве **Devices tree** ниже устройства, которое Вы обновили, также автоматически обновляются.

Параметры конфигурации устройства не изменяются, если тип устройства не был изменен.

Если процедура обновления вызывает какое-либо несоответствие в существующей конфигурации, это обнаруживается при следующем выполнении **Build (**сборки) приложения. Обнаруженные несоответствия обозначаются соответствующими сообщениями. Это также касается неявно добавленных библиотек, которые не будут удалены автоматически и соответственно при обновлении устройства.

Раздел 5.4

Преобразование устройств

Преобразование устройств

Введение

SoMachine 4.0 и более поздние версии позволяют Вам преобразовывать устройство, которое сконфигурировано в Вашем проекте к другому, но совместимому устройству. SoMachine автоматически преобразовывает текущую конфигурацию устройства в выбранное устройство и отображает изменения, которые внесены в виде сообщений **Messages**.

Команда **Convert Device** может автоматически добавить или удалить модули. Эти аппаратные изменения также имеют влияния на адресацию и библиотеки.

Чтобы избежать непреднамеренного поведения после преобразования устройства:

- Проверьте, что новое устройство поддерживает все функции и коммуникационные порты, которые требуются в Вашем проекте.
- Избегайте использования прямых адресов в своем приложении.
- Выполните резервное копирование проекта на жесткий диск PC прежде, чем преобразовать устройство.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННАЯ РАБОТА ОБОРУДОВАНИЯ

- Проверьте, что любые прямые адреса, используемые в Вашем приложении (например, `%IB5`), были преобразованы правильно после преобразования устройства.
- Проверьте, что измененный проект содержит намеченные конфигурации и обеспечивает намеченную функциональность после того, как Вы преобразовали устройство.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

ПРИМЕЧАНИЕ

ПОТЕРЯ ДАННЫХ

Выполните резервное копирование проекта на жесткий диск PC прежде, чем преобразовать устройство.

Отказ следовать этим инструкциям может привести к повреждению оборудования.

Преобразование устройства

Чтобы преобразовать устройство в совместимое устройство, выполните следующее:

Шаг	Действие
1	Выполните резервное копирование проекта на жесткий диск PC, выполняя команду **File** → **Save Project As...** прежде, чем преобразовать устройство.
2	Щелкните правой кнопкой по устройству, которое Вы хотите преобразовать в дереве **Devices Tree**.
3	Выберите команду **Convert Device** из контекстного меню. **Результат**: отображается диалоговое окно **Convert Device**. Оно перечисляет те устройства, которые являются совместимыми с устройством, которое Вы выбрали и предоставляете дополнительную информацию о выбранном устройстве:
4	Выберите из списка устройство, в которое Вы хотите преобразовать свое сконфигурированное в настоящее время устройство. Чтобы отобразить доступные версии устройства, выберите опцию **Display all versions (**все версии только для экспертов)
5	Если Вы еще не выполнили резервное копирование своего проекта, нажмите **Cancel**, чтобы остановиться без изменений и выполнить резервное копирование, прежде чем Вы запустите процедуру еще раз. Чтобы запустить преобразование, нажмите **OK**. **Результат:** сконфигурированное в настоящее время устройство преобразовано в устройство, которое Вы выбрали из списка. Информация, которую Вы вводили, сохранена, если связанные модули все еще доступны. Любые модификации или конфигурации, которые не могли быть преобразованы, перечислены в виде сообщений **Messages**.
6	Проверьте, содержит ли переделанный проект всё еще намеченные конфигурации и обеспечивает намеченные функции. В противном случае адаптируйте конфигурацию или восстановите резервное копирование неизменного файла проекта.

Информация о преобразовании в виде Messages

Следующая информация отображается в виде сообщений **Messages** для процесса преобразования:

- исходные устройства и целевые устройства, в которые они были преобразованы
- параметры, которые не были переданы целевым устройствам
- устройства, которые не были преобразованы

Для сохранения информации отображённой в виде **Messages**, Вы можете скопировать её в буфер обмена (нажмите CTRL + C), и вставьте её в файл данных (нажмите CTRL + V).

Особый случай: преобразование устройства HMI к другому устройству HMI с более высоким разрешением экрана

Как и другие устройства, Вы можете также преобразовать устройство HMI в другое устройство HMI. В этом случае диалоговое окно **Convert Device** включает дополнительную опцию для устройств HMI, которая позволяет автоматическую адаптацию к более высокому разрешению экрана.

Диалоговое окно **Convert Device** для устройств HMI

Если у нового устройства HMI есть больший экран и, таким образом, более высокое разрешение экрана, опция **Adapt to increased screen resolution** включена по умолчанию. Она автоматически адаптирует содержимое панелей HMI и всплывающих окон, а также шрифтов панелей HMI к увеличенному разрешению экрана нового устройства HMI.

ПРИМЕЧАНИЕ: Этот процесс не может быть отменен автоматически. Проверьте и, при необходимости, вручную адаптируйте содержимое панелей после преобразования.

Раздел 5.5

Преобразование проектов

Преобразование проектов SoMachine Basic и Twido

Введение

С SoMachine Вы можете преобразовать проект SoMachine Basic или TwidoSoft/TwidoSuite и сконфигурированный контроллер к выбираемому логическому или HMI контроллеру SoMachine *(см. стр. 879)*. Контроллер и соответствующая логика преобразуются и интегрируются в проект SoMachine. SoMachine обеспечивает различные способы выполнения этого процесса преобразования:

- В SoMachine Central выполните команду **Convert...** → **Convert Twido Project...** или **Convert...** → **Convert SoMachine Basic Project...** из Главного меню *(см. SoMachine Central, User Guide)*. Открывается диалоговое окно **Convert SoMachine Basic Project** или **Convert Twido Project**.
- В SoMachine Logic Builder выполните команду **File** → **Convert SoMachine Basic Project** или **File** → **Convert Twido Project**. Открываются диалоговое окно **Convert SoMachine Basic Project** или диалоговое окно **Convert Twido Project**. Если команды не доступны, Вы можете вставить их в меню по Вашему выбору при помощи диалогового окна **Tools** → **Customize** *(см. SoMachine, Menu Commands, Online Help)*.
- В SoMachine Central откройте проект Twido или SoMachine Basic. Чтобы достичь этого, нажмите кнопку **Open an existing project** или нажмите **Open Project** на экране **Get started** для открытия диалогового окна **Open project**. Выберите опцию **Twido Project files (*.xpr, *.twd, *.xar)** или **SoMachine Basic Project files (*.smbp)** из списка **All supported files** и просмотрите Ваши проекты SoMachine Basic или Twido. Нажмите кнопку **Open**, чтобы открыть диалоговое окно **Convert SoMachine Basic Project** или диалоговое окно **Convert Twido Project**.

Версии SoMachine Basic, поддерживаемые этим механизмом преобразования, перечислены в информации о версии SoMachine. Если Вы преобразовываете проект SoMachine Basic, который создавался с Основной версии SoMachine, которая является более новой, чем последняя поддерживаемая версия, это обозначается сообщением в виде **Messages** *(см. SoMachine, Menu Commands, Online Help)*. Тогда Вы можете продолжить или отменить преобразование. Если Вы будете продолжать, приложение будет преобразовано, но может быть невозможно сделать это без ошибок, которые должны быть исправлены. В этом случае рассмотрите и проверьте вид сообщений и Ваше приложение, прежде чем попытаетесь ввести его в эксплуатацию.

ПРИМЕЧАНИЕ: Проверьте, что проект SoMachine Basic или Twido верен, прежде чем Вы преобразуете его в SoMachine.

Чтобы помочь избежать непреднамеренного поведения после преобразования проекта, проверьте, что целевой контроллер поддерживает функции и коммуникационные порты, которые требуются в Вашем проекте.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННАЯ РАБОТА ОБОРУДОВАНИЯ

- Проверьте, что программа для целевого контроллера содержит намеченные конфигурации и обеспечивает намеченные функции после того, как Вы преобразовали проект.
- Полностью отладьте, верифицируйте и проверьте функциональность переделанной программы прежде, чем ввести ее в эксплуатацию.
- Прежде, чем преобразовать программу, проверьте, что исходная программа верна, т.е. загружаема в исходный контроллер.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования

ПРИМЕЧАНИЕ: Для получения дополнительной информации, совета и важной информации о безопасности относительно импорта проектов в SoMachine см. *SoMachine Compatibility and Migration User Guide (Руководство пользователя по совместимости SoMachine и миграции – SoMachine Compatibility and Migration, User Guide*).

Преобразование проекта SoMachine Basic или Twido

Чтобы преобразовать проект SoMachine Basic или Twido, выполните следующее:

Шаг	Действие
1	Для запуска процесса преобразования, выполните одно из трех действий в SoMachine Central или в SoMachine Logic Builder (как упомянуто в разделе *Введение* этой главы *(см. стр. 77)*). **Результат**: открывается диалоговое окно **Convert SoMachine Basic Project** или диалоговое окно **Convert Twido Project**:
2	Введите имя контроллера в поле **Device Name**.
3	Введите путь к файлу проекта SoMachine Basic или Twido в поле **Project File** или щелкните кнопку **...**, чтобы просмотреть файл. **ПРИМЕЧАНИЕ**: Если Вы уже нашли свой проект SoMachine Basic или Twido в открытом диалоговом окне **Open project**, путь будет введён в поле **Project File** автоматически и не будет редактироваться.

Шаг	Действие
4	Выберите язык программирования, в который логика будет преобразована из списка **Implementation Language (**языка реализации). Поддерживаются следующие языки программирования: ● Лестничная схема (LD) ● Схема функциональных блоков (FBD) ● Список инструкции (IL) ● Непрерывная схема функций (CFC)
5	Выберите целевой контроллер из списка **Devices**, в который Вы хотите преобразовать свой контроллер SoMachine Basic или Twido. Дополнительная информация о выбранном устройстве отображается в информационной области **Information** диалогового окна.
6	Нажмите **Convert**, чтобы запустить преобразование. **Результат:** проект SoMachine Basic или Twido преобразован и интегрирован в открытый проект SoMachine. Модификации или конфигурации, которые не могли быть преобразованы, перечислены в виде сообщений **Messages** *(см. SoMachine, Menu Commands, Online Help)*.
7	Проконсультируйтесь категорией **Project Conversion** вида **Messages** и проверьте обнаруженные и перечисленные ошибки и предупреждения.
8	Проверьте, содержит ли преобразованный проект намеченные конфигурации и обеспечивает ли намеченные функции. В противном случае адаптируйте конфигурацию.

IEC совместимость имен объектов и переменных

Имена объектов и переменных в проектах SoMachine должны соответствовать соглашениям о присвоении имен, определенным в стандарте IEC. Любые имена в Вашем проекте SoMachine Basic или Twido, которые не соответствуют стандарту, автоматически адаптируются преобразователем к соглашениям IEC.
Если Вы хотите сохранить имена, которые не являются совместимыми с IEC в преобразованном проекте SoMachine, активируйте опцию **Allow unicode characters for identifiers (**Разрешить символы юникода для идентификаторов) в диалоговом окне **Project Settings** → **Compile options** *(см. SoMachine, Menu Commands, Online Help)*.

Библиотека TwidoEmulationSupport

Библиотека TwidoEmulationSupport *(см. Twido Emulation Support Library, Library Guide)* содержит функции и функциональные блоки, которые предоставляют функциональность SoMachine Basic и TwidoSoft/TwidoSuite в приложении SoMachine. Библиотека TwidoEmulationSupport автоматически интегрирована в проекте SoMachine с преобразованным контроллером.

Преобразование прикладной программы

В целевом проекте SoMachine отдельные программы создаются для каждой POU SoMachine Basic и свободной POU и для каждой подпрограммы Twido и секции программы. Язык программирования, который используется для этих программ, определен **Implementation Language**, выбранном в диалоговом окне **Convert SoMachine Basic Project / Convert Twido Project**. Исключение сделано для POU, которые были запрограммированы в графическом языке Grafcet. Они преобразованы в программу SFC. Для получения дальнейшей информации обратитесь к разделу Grafcet в этой главе *(см. стр. 87)*.

Для каждого объекта языка (такого, как объекты памяти или функциональные блоки) используемого прикладной программой, создается одна глобальная переменная. Создаются отдельные списки глобальных переменных *(см. стр. 195)* для различных категорий объектов (один для битов памяти, один для слов памяти и т.д.).
Следующие ограничения применяются к преобразованию прикладной программы относительно структуры программы:

- В SoMachine невозможно перейти к метке *(см. стр. 284)* в другой программе.
- Невозможно определить шаги Grafcet в подпрограмме.
- Невозможно активировать или деактивировать шаги Grafcet (через # и D# инструкцию) в подпрограмме.

Преобразование объектов памяти

Области, предусмотренные для объектов памяти в SoMachine Basic и Twido, отличаются от SoMachine. В SoMachine Basic и Twido есть три отличных области для объектов памяти:

Область	Содержащиеся объекты памяти
область битовой памяти	биты памяти (`%M`)
область памяти слов	• слова памяти (`%MW`) • двойные слова (`%MD`) • значения с плавающей точкой (`%MF`)
область констант	• слова констант (`%KW`) • двойные слова констант (`%KD`) • значения констант с плавающей точкой (`%KF`)

В SoMachine есть только область слов памяти для объектов памяти:

Область	Содержащиеся объекты памяти
область памяти слов	• слова памяти (`%MW`) • двойные слова (`%MD`) • значения с плавающей точкой Нет никакого специального формата адресации для значений с плавающей точкой. Переменные с плавающей точкой могут быть размещены по адресам %MD.

Диаграмма обеспечивает обзор различных разметок адресов %MD и %MF в SoMachine Basic / Twido и SoMachine.

1 Адреса памяти в SoMachine Basic / Twido
2 Адреса памяти в SoMachine

Объекты памяти преобразовываются следующим образом:

Исходные объекты памяти	Целевой объект памяти	Дополнительная информация
%MW	Размещается в том же %MW **Пример** %MW2 размещён в %MW2.	Для каждого объекта %MW создается глобальная переменная типа INT.
%MD и %MF с чётными адресами	Размещаются так, что они расположены на одном и том же адресе перед прежним %MW. **Пример** %MD4 / %MF4 размещён в %MD2.	Для каждого объекта %MD создается глобальная переменная типа DINT. Для каждого использованного объекта %MF, создается глобальная переменная типа REAL.
%MD и %MF с нечётными адресами	Не может быть размещён, потому что переменная DINT не может быть расположена на нечетном адресе слова.	Переменная создается, чтобы помочь гарантировать, что преобразованное приложение может быть собрано. Однако Вы должны исследовать эффект, который такое создание переменное имеет на полную функциональность Вашей программы.
%M	Размещается как упакованное битовое поле в фиксированном положении в области %MW.	Для каждого объекта %M создается глобальная переменная типа BOOL.
%KW	Размещается в последовательных адресах области %MW.	Для каждого объекта %KW создается глобальная переменная типа INT.

Отношение между объектами %KW, %KD и %KF такое же, как для объектов %MW, %MD и %MF. Например, %KD4 / %KF4 размещены в том же месте, что и %KW4. Нечетные адреса %KD / %KF не могут быть размещены.

Удаленный доступ

К объектам памяти (%MW, %MD, %MF, и %M) можно получить доступ от удаленного устройства через сервисы Modbus:

- Если удаленное устройство имело доступ к объектам %MW, %MD или %MF в исходном приложении, этот доступ также будет доступен в приложении SoMachine.
- Если удаленное устройство имело доступ к объектам %M в исходном приложении, этот доступ больше не будет доступен в приложении SoMachine.

Преобразование функциональных блоков

Для следующих функциональных блоков в SoMachine Basic / Twido библиотека TwidoEmulationSupport предоставляет функциональные блоки с совместимыми функциями:

Функциональный блок SoMachine Basic / Twido	Функциональный блок библиотеки TwidoEmulationSupport
Timers %TM (Таймеры)	FB_Timer
Counters %C (Счетчики)	FB_Counter
Register %R (Регистр)	FB_FiFo / FB_LiFo
Drum %DR (Барабан)	FB_Drum
Shift bit register %SBR (Сдвиговый регистр)	FB_ShiftBitRegister
Step counter %SC (Счетчик шагов)	FB_StepCounter
Schedule %SCH (Расписание)	FB_ScheduleBlock
PID (ПИД)	FB_PID
Exchange / message %MSG (Обмен/сообщение)	FB_EXCH
High-speed counter %HSC / %VFC (Высокоскоростной счетчик)	Они преобразовываются, как описано в разделе *Преобразование быстрых счётчиков, высокоскоростных счетчиков (Twido: Очень быстрых счетчиков) и импульсных генераторов (см. стр. 86)* этой главы.
Fast counter %FC (Быстрый счетчик)	
PLS pulse generator %PLS (Импульсный генератор)	
PWM pulse generator %PWM (Генератор ШИМ)	
PTO function blocks %PTO, %MC_xxx	
Frequency generator %FREQGEN (Генератор частоты)	
Communication function blocks READ_VAR, WRITE_VAR, WRITE_READ_VAR, и SEND_RECV_MSG (Коммуникационные блоки)	FB_ReadVar, FB_WriteVar, FB_WriteReadVar, и FB_SendRecvMsg
SMS function block SEND_RECV_SMS Блок SMS функций	Они не преобразовываются
MC_MotionTask_PTO	

Для преобразования функциональных блоков выполните следующее:

- Библиотека TwidoEmulationSupport не обеспечивает функциональных блоков для связанных с аппаратными средствами функций, таких как высокоскоростные счетчики, быстрые счетчики и импульсные генераторы. Ими нужно управлять через функциональные блоки, обеспеченные специфическими для платформ библиотеками HSC и PTO_PWM. Эти функциональные блоки несовместимы с исходными функциональными блоками. Короче говоря, полное преобразование невозможно, если исходная программа содержит функции на основе аппаратных ресурсов контроллера. Для получения дополнительной информации обратитесь к описанию *Преобразование быстрых счётчиков, высокоскоростных счётчиков (Twido: Very Fast Counters) и импульсных генераторов* *(см. стр. 86)*.
- В SoMachine Basic / Twido, функция обмена сообщениями обеспечена инструкцией `EXCHx` и функциональным блоком `%MSGx`. В приложении SoMachine эта функция выполнена единственным функциональным блоком `FB_EXCH`.
- В SoMachine Basic / Twido, определенные функциональные блоки могут быть сконфигурированы с использованием специальных диалоговых окон конфигурации. Эти конфигурации обеспечены для функциональных блоков библиотеки TwidoEmulationSupport специальными параметрами.

Преобразование системных переменных

Преобразовываются следующие системные биты и слова:

Системный бит/слово	Дополнительная информация
`%S0`	Установлен в 1 в первом цикле после "холодного" запуска. **ПРИМЕЧАНИЕ:** не возможно инициировать "холодный" запуск записью в этот системный бит.
`%S1`	**ПРИМЕЧАНИЕ:** не возможно инициировать "теплый" запуск записью в этот системный бит.
`%S4`	Импульс с базой времени 10 мс.
`%S5`	Импульс с базой времени 100 мс.
`%S6`	Импульс с базой времени 1 с.
`%S7`	Импульс с базой времени 1 минута.
`%S13`	Устанавливается в 1 в первом цикле после того, как контроллер был запущен.
`%S18`	Устанавливается в 1, если происходит арифметическое переполнение. **ПРИМЕЧАНИЕ:** Этот флаг обеспечен библиотекой TwidoEmulationSupport и установлен только функциями, обеспеченными этой библиотекой.
`%S21` , `%S22`	Только запись. Чтение для этих переменных не поддерживается.
`%SW63...65`	Код ошибки блоков `MSG` 1...3.
`%SW114`	Разрешает флаги для блоков расписания.

Другие системные переменные преобразованием не поддерживаются. Если неподдерживаемая системная переменная используется программой исходного приложения, сообщение сгенерировано в категории **Project Conversion** вида **Messages** *(см. SoMachine, Menu Commands, Online Help)*.

Преобразование сохранения поведения

Переменные и функциональные блоки в SoMachine Basic / Twido сохраняют переменные. Это означает, что они сохраняют свои значения и состояния даже после непредвиденного завершения работы контроллера, а также после нормального цикла отключения и включения питания контроллера.

Это сохранение поведения не сохраняется во время преобразования. В SoMachine преобразованные переменные и функциональные блоки однообразные, что означает, что они инициализируются во время непредвиденного завершения работы и цикла отключения и включения питания контроллера. Если Вы должны сохранить переменные в своем приложении SoMachine, Вы должны объявить это ключевое слово атрибута *(см. стр. 518)* вручную.

Преобразование таблиц анимации

Управление таблицами анимации отличается в исходном и целевом приложениях:

- SoMachine Basic / Twido позволяет Вам определять многочисленные списки анимации, идентифицированные по имени. Каждый список анимации может содержать многочисленные поля для объектов, которые будут анимированы. Для каждой переменной Вы можете выбрать опцию **Trace**.
- В SoMachine есть 4 предопределенных списка наблюдения *(см. стр. 416)* (**Watch 1... Watch 4**). Каждый список наблюдения может содержать многочисленные переменные, которые будут анимированы. Один список наблюдения может содержать переменные из различных контроллеров.

Для тех переменных, которые имеют опцию **Trace** в SoMachine Basic / Twido, SoMachine создает объект трассировки. Вы можете просмотреть эти переменные в редакторе трассировки *(см. стр. 449)*.

Во время процесса преобразования поля исходных анимационных таблиц добавляются в конце списка просмотра **Watch 1**.

Преобразование символов

Символы, определенные в проекте SoMachine Basic / Twido, автоматически передаются в проект SoMachine.

К наименованию символов применяются следующие ограничения:

Если...	Тогда ...
имя символа не соответствует правилам наименования SoMachine,	имя символа изменяется.
имя символа совпадает с ключевым словом SoMachine,	имя символа изменяется.
не создается переменная для объекта языка,	имя символа отбрасывается.
символ нигде не используется в прикладной программе,	имя символа может быть отброшено.

Для полного списка модификаций символов, которые требовались, обратитесь к виду **Messages**.

Преобразование Fast Counters, High-Speed Counters (Twido: Very Fast Counters) и Pulse Generators

Функциональные блоки, обеспечиваемые SoMachine, отличаются от функциональных блоков, обеспеченных SoMachine Basic / Twido. Тем не менее, конфигурация быстрых счетчиков, высокоскоростных счетчиков и импульсных генераторов преобразовывается в максимально возможной степени. Следующие разделы обеспечивают обзор ограничений, которые применяются.

Общие ограничения

Применяются следующие общие ограничения:

Ограничение	Решение
Входы и выходы, используемые преобразованными высокоскоростными счетчиками и импульсными генераторами, могут отличаться от используемых входов и выходов исходного приложения	Примите это во внимание в соединении проводов для преобразованного контроллера. О переназначении входов и выходов сообщается в виде **Messages** *(см. SoMachine, Menu Commands, Online Help)*.
SoMachine Basic поддерживает до четырех высокоскоростных счетчиков и до четырех импульсных генераторов. Функция преобразования преобразует только количество счетчиков и импульсных генераторов, которые поддерживаются целевым контроллером.	Вы должны адаптировать свое приложение вручную.

Ограничения, применяющиеся к `%FC`, `%HSC` / `%VFC`, `%PLS` и `%PWM`

Для каждого функционального блока `%FC`, `%HSC` / `%VFC`, `%PLS` и `%PWM`, используемого в приложении SoMachine Basic / Twido, в SoMachine создается единственная программа. Вы можете улучшить эту основную реализацию согласно потребностям Вашего приложения.

Применяются следующие ограничения:

Ограничение	Решение
Некоторые комбинации счетчиков `%HSC` / `%VFC` и `%FC` не могут быть преобразованы, потому что они не поддерживаются целевым контроллером. Например, в SoMachine Basic можно иметь 2 `%HSC` функциональных блока и 4 `%FC` функциональных блока параллельно. Функция преобразования преобразует только количество функциональных блоков, которое поддерживается целевым контроллером.	Вы должны адаптировать свое приложение вручную.
Доступ к параметрам функционального блока выполняется по-разному в SoMachine Basic и SoMachine. В SoMachine Basic к параметрам функционального блока может получить доступ непосредственно прикладная программа, например, `%VFC.P = 100`. В SoMachine специфичный для контроллера функциональный блок (например, `EXPERTSetParam`) должен использоваться, чтобы получить доступ к параметру.	Если исходное приложение имеет доступы к параметрам функционального блока, Вы должны расширить соответственно преобразованное приложение.

Ограничение	Решение
Поведение счетчиков в SoMachine отличается от Twido, когда устанавливается предварительно заданное значение. В Twido: ● Обратный счетчик продолжает считать, если достигнут нуль. ● Суммирующий счетчик продолжает считать, если достигнуто предварительно заданное значение. В SoMachine: ● Обратный счетчик прекращает считать, если достигнут нуль. ● Суммирующий счетчик начинает считать с начала, если достигнуто предварительно заданное значение.	Вы должны адаптировать свое приложение вручную.

Ограничения, применяющиеся к функциональным блокам `PTO %PTO, %MC_xxxx`

Для M241:

Функциональные блоки PTO, обеспеченные SoMachine для контроллеров M241, совместимы с функциональными блоками PTO, обеспеченными в SoMachine Basic. Функциональные блоки PTO преобразовываются без ограничений. Единственное исключение - функциональный блок `MC_MotionTask_PTO`. Блок `MC_Motion-Task_PTO` не преобразовывается.

Для HMI SCU:

Функциональные блоки PTO, обеспеченные SoMachine для контроллеров HMI SCU, не совместимы с функциональными блоками PTO, обеспеченными в SoMachine Basic. Функциональные блоки PTO не преобразовываются.

Ограничения, применяющиеся к функциональному блоку генератора частоты `%FREQGEN`

Функциональный блок генератора частоты `%FREQGEN` преобразовывается без ограничений для контроллеров и M241 и HMI SCU.

Преобразование программы Grafcet

Вы можете записать программу Grafcet текстовым или графическим способом.

Тип Grafcet	Описание	Поддерживается
Текстовый	Различные программные элементы IL и LD доступны для определения, активации и деактивации состояний Grafcet.	● TwidoSoft/TwidoSuite ● SoMachine Basic
Графический	Позволяет Вам рисовать расположение шагов, переходы и ответвления графическим способом.	Только SoMachine Basic V1.4 и более поздние версии.

Преобразование текстового Grafcet

Языки программирования SoMachine не поддерживают программирование с Grafcet.

По этой причине переделанное приложение Grafcet содержит дополнительные элементы языка, которые реализуют управление Grafcet.

Дополнительный элемент	Описание
папка **Grafcet**	Эта папка содержит следующие элементы языка, используемые для управления автоматом состояния Grafcet.
структура данных `GRAFCET_STATES`	У этой структуры данных есть однобитовый элемент для каждого разрешённого состояния Grafcet. Если это - начальное состояние, элемент инициализирован в TRUE (1), иначе он равен - FALSE (0).
список глобальных переменных **GrafcetVariables**	Этот список глобальных переменных содержит следующие переменные: ● 1 переменная `STATES`, которая содержит 1 бит для каждого состояния Grafcet. Каждый бит представляет текущее значение соответствующего состояния Grafcet (`%Xi` объект). ● 1 переменная `ACTIVATE_STATES`, которая содержит 1 бит для каждого состояния Grafcet. Если бит TRUE, Grafcet активируется в следующем цикле. ● 1 переменная `DEACTIVATE_STATES`, которая содержит 1 бит для каждого состояния Grafcet. Если бит TRUE, состояние Grafcet деактивируется в следующем цикле.
программа **Grafcet**	Эта программа реализует конечный автомат Grafcet. Она содержит логику для активации и деактивации шагов Grafcet. Программа содержит следующие действия: ● `Init` инициализирует шаги Grafcet к их начальным состояниям. Это выполняется, когда системный бит `%S21` установлен прикладной программой. ● `Reset` сбрасывает шаги Grafcet к FALSE. Это выполняется, когда системный `%S22` бит установлен прикладной программой.

Инструкции Grafcet в прикладной программе преобразовываются следующим образом:

- Начало каждого шага Grafcet отмечено меткой с именем шага.
  Первый оператор в шаге Grafcet проверяет, активен ли шаг. В противном случае он переходит к метке следующего шага Grafcet.
- Доступ к `%Xi` преобразуется в доступ к переменной `STATES.Xi`.
- Инструкция активации Grafcet `#i` преобразуется в установку бита активации состояния `i` и деактивации бита текущего состояния.
- Инструкция деактивации Grafcet `#Di` преобразована в установку бита деактивации состояния `i` и деактивации бита текущего состояния.

Вы можете расширить преобразованную программу Grafcet, если Вы рассмотрели информацию, приведенную в этом разделе.

Преобразование графического Grafcet

Графический Grafcet подобен языку программирования SFC, обеспеченного SoMachine. Поэтому POU графического Grafcet преобразовывается в программу SFC в максимально возможной степени.

Есть следующие различия между графическим Grafcet и SFC:

Графический Grafcet	SFC	Дополнительная информация
Может иметь произвольное число начальных шагов.	Должен иметь точно один начальный шаг.	Если у графического Grafcet POU есть несколько начальных шагов, то преобразователь создает несколько начальных шагов в SFC. Это приводит к тому, что преобразованное приложение не может быть создано без обнаруживаемых ошибок. Тщательно адаптируйте преобразованную программу.
Разрешена активация многократных шагов альтернативного ответвления.	Только один шаг альтернативного ответвления может быть активирован.	Тщательно проверьте, что преобразованная программа работает так, как ожидалось.
Выходные переходы шага оцениваются сразу после того, как шаг был выполнен.	Переходы программы SFC оцениваются после того, как все активные шаги были выполнены.	Тщательно проверьте, что преобразованная программа работает так, как ожидалось.
Расположения шагов, переходов и ответвлений относительно свободны.	Расположение шагов, переходы и ответвления более ограничены.	Графическое расположение преобразовано в SFC в максимально возможной степени. О несовместимостях, с которыми встречаются во время преобразования, сообщают в виде **Messages**. Действия шагов и секции переходов полностью преобразовываются. Завершите созданную SFC по мере необходимости.

POU графического Grafcet может быть инициализирована установкой системного бита `%S21`. Если этот бит устанавливается в проекте SoMachine Basic, преобразователь активирует неявную переменную `SFCInit` и использует её, чтобы инициализировать программу SFC.

Преобразование модулей расширения TM2 к модулям расширения TM3

Контроллеры Twido используют только модули расширения TM2. Даже притом, что логические контроллеры M221 и M241 могут поддерживать модули как TM2, так и TM3, наиболее успешная практика – использовать модули TM3. Чтобы преобразовать модули TM2, используемые в Вашем проекте Twido, в модули TM3 для проекта SoMachine, по умолчанию выбирается опция **Upgrade TM2 Modules to TM3**.

Модули расширения TM2 преобразовываются в модули расширения TM3, как перечислено в таблице:

Исходный модуль расширения TM2	Целевой модуль расширения TM3	Дополнительная информация
TM2DDI8DT	TM3DI8	–
TM2DAI8DT	TM3DI8A	–
TM2DDO8UT	TM3DQ8U	–
TM2DDO8TT	TM3DQ8T	–

Исходный модуль расширения TM2	Целевой модуль расширения TM3	Дополнительная информация
TM2DRA8RT	TM3DQ8R	–
TM2DDI16DT	TM3DI16	–
TM2DDI16DK	TM3DI16K	–
TM2DRA16RT	TM3DQ16R	–
TM2DDO16UK	TM3DQ16UK	–
TM2DDO16TK	TM3DQ16TK	–
TM2DDI32DK	TM3DI32K	–
TM2DDO32UK	TM3DQ32UK	–
TM2DDO32TK	TM3DQ32TK	–
TM2DMM8DRT	TM3DM8R	–
TM2DMM24DRF	TM3DM24R	–
TM2AMI2HT	TM3AI2H	–
TM2AMI4LT	TM3TI4	Возможно, что поведение преобразованного температурного модуля отличается от исходного модуля. Тщательно проверьте преобразованный модуль.
TM2AMI8HT	TM3AI8	–
TM2ARI8HT	–	Модули TM2 TM2ARI8HT, TM2ARI8LRJ, и TM2ARI8LT не преобразовываются, потому что нет никакого соответствующего модуля расширения TM3. Вы можете заменить этот модуль двумя модулями TM3TI4.
TM2AMO1HT	TM3AQ2	У целевого модуля расширения TM3 есть больше каналов ввода-вывода, чем у исходного модуля TM2.
TM2AVO2HT		–
TM2AMM3HT	TM3TM3	–
TM2ALM3LT		Возможно, что поведение преобразованного температурного модуля отличается от исходного модуля. Тщательно проверьте преобразованный модуль.
TM2AMI2LT	TM3TI4	У целевого модуля расширения TM3 есть больше каналов ввода-вывода, чем у исходного модуля TM2. Возможно, что поведение преобразованного температурного модуля отличается от исходного модуля. Тщательно проверьте преобразованный модуль
TM2AMM6HT	TM3AM6	–
TM2ARI8LRJ	–	Модули TM2, TM2ARI8LRJ и TM2ARI8LT не преобразованы, потому что нет никакого соответствующего модуля расширения TM3. Вы можете заменить этот модуль двумя модулями TM3TI4.

Исходный модуль расширения TM2	Целевой модуль расширения TM3	Дополнительная информация
TM2ARI8LT	–	Модули TM2 TM2ARI8HT, TM2ARI8LRJ и TM2ARI8LT не преобразованы, потому что нет никакого соответствующего модуля расширения TM3. Вы можете заменить этот модуль двумя модулями TM3TI4.

ПРИМЕЧАНИЕ: Если Вы используете модули расширения TM2, а также TM3 в Вашем проекте SoMachine, обратите внимание на их позицию в древовидной структуре: Если узлы TM3 расположены ниже узлов TM2 в древовидной структуре, об этом сообщается как обнаруженной ошибке **Build** в виде **Messages**.

Непосредственный доступ к Вх/Вых

Инструкции `READ_IMM_IN` и `WRITE_IMM_OUT` SoMachine Basic для немедленного доступа к цифровым локальным каналам Вх/Вых, не преобразовываются.

Для контроллеров HMI SCU не существует эквивалентной функции.

Для контроллеров M241 Вы можете использовать функции `GetImmediateFastInput` и `PhysicalWriteFastOutputs`, предоставленные библиотекой PLCSystem, но рассмотреть следующие различия:

Инструкции `READ_IMM_IN` и `WRITE_IMM_OUT` (контроллеры M221)	Функции `GetImmediateFastInput` и `PhysicalWriteFastOutputs` (контроллеры M241)
Доступ ко всем локальным входам и выходам.	Доступ только к быстрым входам и выходам.
`WRITE_IMM_OUT` пишет один бит (подобно функции чтения). `WRITE_IMM_OUT` возвращает код ошибки.	`PhysicalWriteFastOutputs` пишет в быстрые выходы одновременно. `PhysicalWriteFastOutputs` только возвращает информацию о том, какие выходы были на самом деле записаны.
Коды ошибки `READ_IMM_IN` и `GetImmediateFastInput` отличаются.	
`READ_IMM_IN` обновляет объект входа (`%I0.x`).	`GetImmediateFastInput` только возвращает значение чтения, но не обновляет канал входа.

Коммуникационные функции Twido

Следующие коммуникационные функции Twido не преобразовываются:

- интерфейс AS
- CANopen
- удаленный канал

Если Вы используете эти коммуникационные функции в своем приложении Twido, Вы должны адаптировать приложение SoMachine вручную.

Во время преобразования одна переменная создается для каждого связанного объекта ввода-вывода, чтобы позволить приложению SoMachine быть успешно собранным. Эти переменные собраны в отдельных списках глобальных переменных. Это помогает Вам в идентификации переменных, которые были заменены.

Обнаруженные ошибки и предупреждения, индицируемые в виде сообщений Messages

Если ошибки или предупреждения обнаружены во время процесса преобразования, отображается окно сообщений, индицируя количество обнаруженных ошибок и предупреждений. Для получения дополнительной информации обращайтесь к категории **Project Conversion** вида **Messages** (см. *SoMachine, Menu Commands, Online Help*). Тщательно проверьте каждую запись, чтобы видеть, должны ли Вы адаптировать свое приложение.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННАЯ РАБОТА ОБОРУДОВАНИЯ

- Проверьте, что программа для целевого контроллера содержит намеченные конфигурации и обеспечивает намеченные функции после того, как Вы преобразовали проект.
- Полностью отладьте, верифицируйте и проверьте функциональность преобразованной программы прежде, чем ввести её в действие.
- Прежде, чем преобразовать программу, проверьте, что исходная программа верна, т.е., загружаема в исходный контроллер.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

- Сообщение **warning** указывает, что процесс преобразования внёс некоторые корректировки, которые, по всей вероятности, не оказывают влияние на функции Вашего приложения.
- Сообщение **error** указывает, что некоторые части приложения не могли быть полностью преобразованы. В этом случае Вы должны адаптировать приложение вручную, чтобы сохранить ту же функциональность в целевом приложении.
- Если прикладная программа использует функциональность, которая не может быть полностью преобразована, преобразователь создает переменные для неподдерживаемых объектов языка. Это позволяет Вам успешно скомпилировать свое приложение. Однако проверьте эту неподдерживаемую функциональность после преобразования.

Чтобы сохранить информацию, отображаемую в виде **Messages**, Вы можете скопировать её в Буфер обмена (нажмите CTRL + C), и вставить её в файл данных (нажмите CTRL + V).

Глава 6

Диалоговые окна общих редакторов устройств

Содержание этой главы

Эта глава содержит следующие темы:

Раздел	Тема	Страница
6.1	Конфигурация устройств	94
6.2	Распределение входов/выходов	135

Раздел 6.1

Конфигурация устройств

Содержание раздела

Этот раздел содержит следующие темы:

Тема	Страница
Общая информация о редакторах устройств	95
Выбор контроллера	97
Коммуникационные параметры настройки	111
Конфигурация	115
Приложения	116
Файлы	118
Регистрация	119
Параметры настройки ПЛК	121
Пользователи и группы	123
Task Deployment – развёртывание задач	133
Статус	134
Информация	134

Общая информация о редакторах устройств

Обзор

Редактор устройства обеспечивает параметры для конфигурации устройства, которое управляется в дереве **Devices tree**.

Чтобы открыть редактор устройств для определенного устройства, сделайте следующее:

- Дважды щелкните по узлу устройства в дереве **Devices tree** или
- Выберите устройство в дереве **Devices tree** и выполните команду **Edit Object** через контекстное меню или с помощью меню **Project**.

Диалоговое окно **Tools** → **Options** → **Device editor** позволяет Вам делать невидимыми общие виды конфигурации устройства *(см. стр. 115)*.

В этой главе описываются основные диалоговые окна редактора устройства. Специфичные для шин диалоговые окна конфигурации описаны отдельно.

Основные диалоговые окна редактора устройства

Заголовок основного диалогового окна состоит из имени устройства, например, **MyPlc**.

В зависимости от типа устройства редактор устройства может обеспечить следующие вкладки:

Вкладка	Описание
Controller Selection Выбор контроллера (*см. стр. 97*)	Конфигурация соединения между системой программирования и программируемым устройством (контроллером). Это - вкладка по умолчанию для SoMachine V4.0 и более поздних версий. SoMachine V3.1 и более ранние версии используют по умолчанию вкладку **Communication Settings**.
Communication Settings Коммуникационные параметры настройки *(см. стр. 111)*	Конфигурация связи между системой программирования и программируемым устройством (контроллером). Это - вкладка по умолчанию для SoMachine V3.1 и более ранних версий. SoMachine V4.0 и более поздние версии используют по умолчанию вкладку **Controller Selection**
Configuration Конфигурация *(см. стр. 115)*	Отображение или конфигурация параметров устройства.
Applications Приложения *(см. стр. 116)*	Список приложений, работающих в контроллере в настоящее время. Обратитесь к описанию в главе *Программа (см. стр. 145)*.
Files (Файлы) *(см. стр.118)*	Конфигурация передачи файлов между хостом и контроллером.
Log (Регистрация) *(см. стр. 119)*	Отображение файла системного журнала контроллера.
PLC settings Параметры настройки PLC *(см. стр. 121)*	Конфигурация: ● применение, отмеченное для обработки входов/выходов ● поведение ввода/вывода в статусе остановки ● опции цикла шины
Users and Groups Пользователи и группы *(см. стр. 872)*	Управление пользователями относительно доступа к устройству во время выполнения.
Access Rights Права доступа *(см. стр. 875)*	Конфигурация прав доступа к объектам во время выполнения и к файлам для конкретных групп пользователей.

Вкладка	Описание
Task Deployment (Развертывание задачи) *(см. стр. 133)*	Отображение входов и выходов, назначенных на определенную задачу - используется для поиска неисправностей.
Status Состояние *(см. стр. 134)*	Определенный для устройства статус и диагностические сообщения.
Information Информация *(см. стр.134)*	Общая информация об устройстве (например: имя, поставщик, версия).
I/O Mapping Распределение устройств ввода-вывода *(см. стр. 135)*	Отображение каналов входа и выхода устройства ввода/вывода на переменные проекта (приложения).

Выбор контроллера

Обзор

Вид **Controller selection** редактора устройства обеспечивает доступ к сервису сетевой идентификации устройства. Это позволяет Вам сканировать сеть Ethernet для доступных устройств (таких как контроллеры, устройства HMI) и отображать их в списке. В этом виде Вы можете формировать параметры для связи между устройствами (называемыми *контроллерами* в этой главе) и системой программирования.

Список контроллеров содержит те контроллеры в сети, которые послали ответ на запрос SoMachine. Может произойти так, что выбранный Вами контроллер не включен в этот список. У этого может быть несколько причин. За объяснением причин и подходящими решениями, обратитесь к главе, *Доступ к контроллерам – выявление ошибок и часто возникающие вопросы (см. стр. 785)*.

Вид **Controller selection** видим, только если связь между контроллером и системой программирования установлена при помощи IP-адреса. Это - настройка по умолчанию для SoMachine V4.0 и более поздних версий. Вы можете выбрать между созданием коммуникации через IP-адрес и через активный путь в диалоговом окне параметров настройки **Project Settings** → **Communication settings**. Если выбрана опция **Dial up via “IP-address”**, вид **Controller selection** отображается в редакторе устройства. Иначе отображается вид **Communication settings**.

Вид **Controller selection (**выбора контроллера) в редакторе устройства.

Вид **Controller selection** обеспечивает следующие элементы:

- кнопки на панели инструментов
- список, предоставляющий информацию о доступных контроллерах
- выбор опции, список и текстовое окно внизу вида.

Описание кнопок на панели инструментов

На панели инструментов доступны следующие кнопки:

Кнопка	Описание
Optical (вспышки)	Нажмите эту кнопку, чтобы заставить выбранный контроллер генерировать оптический сигнал: это вызывает быстрое вспыхивание светодиода контроля. Это может помочь Вам опознать данный контроллер, если используются много контроллеров. Функция останавливается по второму щелчку или автоматически примерно через 30 секунд. **ПРИМЕЧАНИЕ**: светодиод работает только у контроллеров, поддерживающих эту функцию.
Optical and acoustical (вспышки и звуки)	Нажмите эту кнопку, чтобы заставить выбранный контроллер генерировать оптический и акустический сигналы: Он начинает сигналить звуком и быстро вспыхивающим светодиодом контроля. Это может помочь Вам опознать соответствующий контроллер, если используются много контроллеров. Функция останавливается по второму щелчку или автоматически примерно через 30 секунд. **ПРИМЕЧАНИЕ**: оптический и акустический сигналы выдаются только контроллерами, которые поддерживают эту функцию.
Update (обновление)	Нажмите эту кнопку, чтобы обновить список контроллеров. Запрос отправляется контроллерам в сети. Любой контроллер, который отвечает на запрос, перечисляется с текущими значениями. Предустановленные записи контроллеров обновляются с каждым новым запросом. Контроллеры, которые уже находятся в списке, но не отвечают на новый запрос, не удаляются. Они помечаются как бездействующие красным крестом, добавляемым к символу контроллера. Кнопка **Update** соответствует команде **Refresh list**, которая присутствует в контекстном меню, если Вы щелкаете правой кнопкой мыши по контроллеру в списке. Чтобы обновить информацию выбранного контроллера, контекстное меню предоставляет команду **Refresh this controller**. Эта команда запрашивает более подробную информацию от выбранного контроллера. **ПРИМЕЧАНИЕ:** Команда **Refresh this controller** может также обновить информацию других контроллеров.
Remove inactive controllers from list. (Удаление неактивных контроллеров из списка)	Контроллеры, которые не отвечают на сетевой скан, отмечаются в списке как бездействующие. Они обозначаются красным крестом, добавляемым к символу контроллера. Нажмите эту кнопку, чтобы удалить одновременно все контроллеры в списке, отмеченные как бездействующие. **ПРИМЕЧАНИЕ**: Контроллер может быть отмечен как бездействующий, даже если дело обстоит не так. Контекстное меню, которое открывается, если Вы щелкаете правой кнопкой мыши по контроллеру в списке, обеспечивает 2 других команды для удаления контроллеров: ● Команда **Remove selected controller from list** позволяет Вам удалять из списка только отобранный контроллер. ● Команда **Remove all controllers from list** позволяет Вам удалять из списка все контроллеры одновременно.
New Favorite... и **Favorite 0** (Новый фаворит и Фаворит 0)	Вы можете использовать **Favorites**, чтобы приспособить выбор контроллеров к Вашим личным требованиям. Это может помочь Вам отслеживать много контроллеров в сети. **Favorites** описывает набор контроллеров, которые распознаются по уникальному идентификатору. Нажмите кнопку (такую как **Favorite 0**), чтобы выбрать или отсеять его. Если Вы не выбрали фаворита, все обнаруженные контроллеры видимы. Вы можете также получить доступ к **Favorites** через контекстное меню. Оно открывается после щелчка правой кнопкой мыши по контроллеру в списке. Переместите курсор на кнопку фаворита на панели инструментов, чтобы просмотреть связанные контроллеры в виде всплывающих пояснений.

Список контроллеров

Список контроллеров посередине вида **Controller selection** редактора устройства перечисляет те контроллеры, которые послали ответ на скан сети. Он предоставляет информацию о каждом контроллере в нескольких колонках. Вы можете приспособить колонки, показанные в списке контроллеров, согласно Вашим индивидуальным требованиям.

Чтобы получить это, щелкните правой кнопкой мыши по заголовку колонки, чтобы открыть диалоговое окно **Process columns**.

Вы можете создать свое собственное расположение этой таблицы. Нажмите **New** и введите имя для своего расположения. Перемещайте колонки из списка **Possible** (возможных) **columns** в список **Current (**текущих) **columns** и наоборот, нажимая горизонтальные кнопки стрелок. Чтобы изменить порядок колонок в **Current columns**, щелкайте по кнопкам стрелок вверх и вниз.

Конфигурирование настройки коммуникационных параметров

Чтобы установить параметры для коммуникации между системой программирования и контроллером, продолжите следующим образом:

Шаг	Действие
1	Выберите контроллер в списке контроллеров.
2	Щелкните правой кнопкой мыши по записи контроллера и выполните команду **Process communication settings...** из контекстного меню. **Результат**: диалоговое окно **Process communication settings...** открывается с текущими параметрами настройки контроллера. **ПРИМЕЧАНИЕ**: Большинство контроллеров обеспечивает параметр (такой как **RemoteAccess**), который помогает предотвратить изменение коммуникационных параметров контроллера.

Шаг	Действие
3	Сконфигурируйте коммуникационные параметры: ● **Boot Mode** ❍ **FIXED**: фиксированный IP-адрес используется согласно значениям, введенным ниже (**IP address**, **Subnet mask**, **Gateway**). ❍ **BOOTP**: IP-адрес получен динамически через BOOTP (протокол загрузки). Значения ниже будут проигнорированы. ❍ **DHCP**: IP--адрес получен динамически через DHCP (dynamic host configuration protocol). Значения ниже будут проигнорированы. **ПРИМЕЧАНИЕ**: Не все устройства поддерживают BOOTP и/или DHCP. ● **IP address** При конфигурировании IP-адреса, обратитесь к сообщению об опасности ниже. Это текстовое окно содержит IP-адрес контроллера. Это - уникальный адрес, который состоит из 4 чисел в диапазоне 0... 255 отделенных точками. IP-адрес должен быть уникальным в этой (под) сети. ● **Subnet mask** Маска подсети определяет сегмент сети, которому принадлежит контроллер. Это - адрес, который состоит из 4 чисел в диапазоне 0... 255 отделенных точками. Обычно только значения 0 и 255 используются для стандартных чисел маски подсети. Однако возможны другие числовые значения. Значение маски подсети обычно - то же самое для всех контроллеров в сети. ● **Gateway** Адрес шлюза - адрес местного IP маршрутизатора, который расположен в той же самой сети, что и контроллер. IP маршрутизатор передает данные к местам назначения за пределами местной сети. Это - адрес, который состоит из 4 чисел в диапазоне 0... 255 отделенных точками. Значение шлюза обычно - то же самое для всех контроллеров в сети ● Чтобы сохранить коммуникационные настройки в контроллере, даже если он перезапущен, активируйте опцию, **Save settings permanently**..
4	Нажмите **OK**, чтобы передать параметры настройки контроллеру.

Тщательно управляйте IP-адресами, потому что каждое устройство в сети требует уникального адреса. Наличие многочисленных устройств с тем же самым IP-адресом может вызвать непреднамеренную операцию Вашей сети и связанного оборудования.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННАЯ РАБОТА ОБОРУДОВАНИЯ

- Проверьте, что у всех устройств есть уникальные адреса.
- Получите свой IP-адрес от Вашего системного администратора.
- Подтвердите, что IP-адрес устройства уникален, прежде чем ввести систему в работу.
- Не назначайте тот же самый IP-адрес ни на какое другое оборудование в сети.
- Обновите IP-адрес в уникальный адрес после клонирования любого приложения, которое включает коммуникации Ethernet.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

Управление фаворитами

Для управления фаворитами в списке контроллеров проделайте следующее:

Шаг	Действие
1	Выберите контроллер в списке контроллеров
2	Щелкните правой кнопкой мыши по контроллеру и выберите одну из команд: ● **New Favorite**, чтобы создать новую группу фаворитов. ● **Favorite n**, чтобы ❍ добавить отобранный контроллер к этому списку фаворитов ❍ удалить отобранный контроллер из этого списка фаворитов ❍ удалить все контроллеры из этого списка фаворитов ❍ выбрать фаворита ❍ переименовать фаворита ❍ удалить фаворита

Опция безопасного онлайн режима

Опция **Secure online mode** заставляет SoMachine показывать сообщение, требующее подтверждения, когда выбрана одна из следующих онлайн команд: **Force values, Login, Multiple download, Release force list, Single cycle, Start, Stop, Write values**. Чтобы отключить безопасный режим онлайн и таким образом подавить показ этого сообщения, снимите флажок с этой опции.

Определение уникальных имён устройств (NodeNames)

Термин **NodeName** (имя узла) использован как синоним для термина название устройства. Поэтому имена узлов также используются, чтобы опознать контроллер после скана сети, управлять ими так же тщательно, как и IP-адресами, и проверить, что каждое имя узла уникально в Вашей сети. Назначение многочисленным устройствам одинаковых имён узлов может вызвать непредсказуемую работу Вашей сети и связанного оборудования.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННАЯ РАБОТА ОБОРУДОВАНИЯ

- Гарантируйте, чтобы у всех устройств был уникальный nodenames.
- Подтвердите, что имя узла устройства уникально, прежде чем поместить систему на службу.
- Не назначайте то же самое имя узла ни на какое другое оборудование в сети.
- Обновите уникальное имя узла после клонирования любого применения, которое включает коммуникации Ethernet.
- Создайте уникальное имя узла для каждого устройства, которое не создает его автоматически, таких как XBT HMI контроллеры.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

В зависимости от типа контроллера автоматическое создание **NodeName** может отличаться по процедуре. Чтобы создать уникальное имя, некоторые контроллеры интегрируют свой IP-адрес, другие используют MAC адрес адаптера Ethernet. В этом случае Вы не должны менять имя.

Другие устройства, такие как XBT HMI контроллеры, не создают уникальное имя устройства автоматически. В этом случае назначьте уникальное имя устройства (**NodeName**) следующим образом:

Шаг	Действие
1	Щелкните правой кнопкой мыши по контроллеру в списке и выполните команду **Change device name...** из контекстного меню. **Результат**: открывается диалоговое окно **Change device name**.
2	В диалоговом окне **Change device name** введите уникальное имя устройства в текстовом окне **New**.
3	Щелкните по кнопке **OK**, чтобы подтвердить ввод. **Результат**: имя устройства, которое Вы ввели, назначено для контроллера и будет показано в колонке **NodeName** списка. **ПРИМЕЧАНИЕ**: Имя устройства и **NodeName** синонимы.

Конкретизация режима соединения

Список **Connection Mode** в нижнем левом углу вида **Controller selection** позволяет Вам для адреса соединения выбирать формат, в котором Вы должны вводить его в поле **Address**.

Поддерживаются следующие форматы:

- Automatic — Автоматический *(см. стр.103)*
- Nodename — Имя узла *(см. стр. 104)*
- IP Address — IP-адрес *(см. стр. 104)*
- Nodename via NAT (Remote TCP) — Имя узла NAT (удаленный TCP) *(см. стр.105)* (NAT = network address translation)
- IP Address via NAT (Remote TCP) — IP-адрес через NAT (удаленный TCP) *(см. стр. 105)*
- Nodename via Gateway — Имя узла через Gateway *(см. стр. 106)*
- IP Address via Gateway — IP-адрес через Gateway *(см. стр. 108)*
- Nodename via MODEM — Имя узла через MODEM *(см. стр. 109)*

ПРИМЕЧАНИЕ: После того, как Вы изменили **Connection Mode**, может потребоваться выполнить процедуру логина дважды, чтобы получить доступ к выбранному контроллеру.

Режим соединения Automatic

Если Вы выбираете опцию **Automatic** из списка **Connection Mode**, Вы можете ввести имя узла, IP-адрес или URL связи (универсальный идентификатор ресурса), чтобы определить **Address**.

ПРИМЕЧАНИЕ: не используйте пробелы в начале или конце **Address**.

Если Вы выбрали другой **Connection Mode** и определили **Address** для этого способа, адрес, который Вы определили, все еще будет доступен в текстовом окне **Address**, если Вы переключитесь на **Connection Mode** → **Automatic**.

Пример:

Выбран режим **Connection Mode** → **Nodename via NAT** (Удаленный TCP) и определены адрес и имя узла (nodename)

Если Вы переключаетесь на **Connection Mode** → **Automatic**, информация преобразована в URL, начинающийся с префикса enodename3://

Если IP-адрес был введен для способа соединения (например, когда был выбран **Connection Mode** → **IP Address**), информация преобразована в URL, начинающийся с префикса `etcp3://`. Например, `etcp3://<Ipaddress>`.

Если для способа соединения был введено имя узла (nodename) (например, когда был выбран **Connection Mode** → **Nodename**), информация преобразована в URL, начинающийся с префикса `enodename3://`. Например, `enodename3://<Nodename>`.

Режим соединения → Nodename

Если Вы выбираете опцию **Nodename** из списка **Connection Mode**, Вы можете войти в nodename контроллера, чтобы определить **Address**. Текстовое окно заполняется автоматически, если Вы дважды щелкаете по контроллеру в списке контроллеров.

Пример: **Nodename:** `MyM238 (10.128.158.106)`

Если контроллер, который Вы выбрали, не обеспечивает nodename, **Connection Mode** автоматически изменяется на **IP Address**, и IP-адрес из списка вводится в текстовое окно **Address**.

ПРИМЕЧАНИЕ: не используйте пробелы в начале или конце **Address**. Не используйте запятые (**,**) в текстовом окне **Address**.

Режим соединения → IP-адрес

Если Вы выбираете опцию **IP Address** из списка **Connection Mode**, Вы можете ввести IP-адрес контроллера, чтобы определить **Address**. Текстовое окно заполнится автоматически, если Вы дважды щелкаете по контроллеру в списке контроллеров.

Пример: **IP Address:** `190.201.100.100`

Если контроллер, который Вы выбрали, не обеспечивает IP-адрес, **Connection Mode** автоматически изменяется на **Nodename**, и nodename из списка вводится в текстовое окно **Address**.

ПРИМЕЧАНИЕ: Введите IP-адрес согласно формату
<Number>.<Number>.<Number>.<Number>

Режим соединения → Nodename через NAT (Remote TCP)

Если Вы выбираете опцию **Nodename via NAT (Удаленный TCP)** из списка **Connection Mode**, Вы можете определить адрес контроллера, который расположен позади NAT маршрутизатора в сети. Введите в nodename контроллера и IP-адрес или имя хоста и порт NAT маршрутизатора.

1 PC / HMI
2 NAT роутер
3 целевое устройство

Пример: **NAT Address/Port:** `10.128.158.106/1105` **Target Nodename:** `MyM238 (10.128.158.106)`

ПРИМЕЧАНИЕ: Вводите верный IP-адрес (формат `<Number>.<Number>.<Number>.<Number>`) или верное имя хоста для **NAT Address**.

Введите используемый порт NAT маршрутизатора. Иначе по умолчанию используется порт 1105.

Не используйте пробелы в начале или конце и не используйте запятые в текстовом окне **Target Nodename**.

Информация, которую Вы вводите, интерпретируется как URL, который создает отдаленный мост – использующий драйвер TCP блока – и затем соединяется, сканируя на контроллер с заданным nodename на локальном шлюзе.

ПРИМЕЧАНИЕ: NAT маршрутизатор может быть расположен на самом целевом контроллере. Вы можете использовать его, чтобы создать TCP мост к самому контроллеру.

Режим соединения → IP-адрес через NAT (Remote TCP)

Если Вы выбираете опцию **IP Address via NAT (Remote TCP)** (NAT = network address translation) из списка **Connection Mode**, Вы можете определить адрес контроллера, который расположен позади NAT маршрутизатора в сети. Введите IP-адрес контроллера и IP-адрес или имя хоста и порт NAT маршрутизатора.

1 PC / HMI
2 NAT роутер
3 целевое устройство

Пример: **NAT Address/Port:** `10.128.154.206/1217` **Target IP Address:** `192.168.1.55`

ПРИМЕЧАНИЕ: Введите верный IP-адрес (формат `<Number>.<Number>.<Number>.<Number>`) или действительное имя хоста для **NAT Address**.

Введите используемый порт NAT маршрутизатора. Иначе по умолчанию используется порт шлюза SoMachine **1217**.

Введите верный IP-адрес (формат `<Number>.<Number>.<Number>.<Number>`) для **Target IP Address**.

Информация, в которую Вы вводите, интерпретируется как URL, который создает удаленный TCP мост – используя блок TCP драйвера – и затем соединяется, сканируя на контроллер с данным nodename на локальном шлюзе. IP-адрес ищется в nodename (таком как `MyController (10.128.154.207)`) или вызывая сервис на каждое сканированное устройство шлюза.

ПРИМЕЧАНИЕ: NAT маршрутизатор может быть расположен на самом целевом контроллере. Вы можете использовать его, чтобы создать TCP мост к контроллеру.

Режим соединения → Nodename через Gateway

Если Вы выбираете опцию **Nodename via Gateway** из списка **Connection Mode**, Вы можете определить адрес контроллера, который располагается позади или близко к маршрутизатору шлюза SoMachine в сети. Введите nodename контроллера, и IP-адрес или имя хоста и порт маршрутизатора шлюза SoMachine.

1 PC / HMI
2 PC / HMI / устройства с установленным шлюзом SoMachine
3 целевое устройство

Пример: **Gateway Address/Port:** `10.128.156.28/1217` **Target Nodename:** `MyM238`

ПРИМЕЧАНИЕ: Введите верный IP-адрес (формат `<Number>.<Number>.<Number>.<Number>`) или верное имя хоста для **Gateway Address/Port:**.

Введите используемый порт маршрутизатора. Иначе по умолчанию используется порт шлюза SoMachine **1217**.

Не используйте пробелы в начале или конце и не используйте запятые в текстовом окне **Target Nodename**.

Информация, которую Вы вводите, интерпретируется как URL. Шлюз (gateway) сканируется на устройство с данным nodename, которое непосредственно подключено к этому шлюзу. Непосредственно связанные средства в топологии шлюза SoMachine это – сам корневой узел или дочерний узел корневого узла.

ПРИМЕЧАНИЕ: шлюз SoMachine может быть расположен на целевом контроллере, PC назначения или собственно локальном PC, делая возможным соединение с устройством, которое не имеет никакого уникального nodename, но размещается в подсети позже сети SoMachine.

Рисунок показывает пример, который позволяет соединение от PCI / HMI к целевому контроллеру 3 (пункт 4 на рисунке) при помощи адреса промежуточного PC2 (пункт 5 в диаграмме), который имеет установленный шлюз SoMachine.

1 промежуточный (hop) PC 1
2 целевой контроллер 1: MyNotUniqueNodename
3 целевой контроллер 2: MyNotUniqueNodename
4 целевой контроллер 3: MyNotUniqueNodename
5 промежуточный (hop) PC 2
6 PC / HMI
7 маршрутизатор – роутер (router)
8 Ethernet

Режим соединения → IP-адрес через Gateway шлюз

Если Вы выбираете опцию **IP Address via Gateway** из списка **Connection Mode**, Вы можете определить адрес контроллера, который размещён позади или близко к маршрутизатору шлюза SoMachine в сети. Введите IP-адрес контроллера, и IP-адрес или имя хоста и порт маршрутизатора шлюза SoMachine.

1 PC / HMI
2 PC / HMI / устройства с установленным шлюзом SoMachine
3 целевое устройство

Пример: **Gateway Address/Port:** `10.128.156.28`**/**`1217` **Target IP Address:** `10.128.156.222`

ПРИМЕЧАНИЕ: Введите верный IP-адрес (формат `<Number>.<Number>.<Number>.<Number>`) или верное имя хоста для **Gateway Address/Port:**..

Введите используемый порт маршрутизатора шлюза. Иначе по умолчанию используется шлюз SoMachine порт 1217.

Введите верный IP-адрес (формат `<Number>.<Number>.<Number>.<Number>`) для **Target IP Address**.

Информация, которую Вы вводите, интерпретируется как URL. Шлюз сканирует на устройство с заданным IP-адресом. IP-адрес ищется в nodename (таком как `MyController (10.128.154.207)`) или вызывая сервис на каждое сканированное устройство шлюза.

ПРИМЕЧАНИЕ: шлюз SoMachine может быть расположен на целевом контроллере, PC назначения или самом локальном PC. Поэтому можно соединиться с устройством, которое не имеет никакого уникального nodename, но размещено в подсети после сети SoMachine.

Рисунок показывает пример, который позволяет соединение от промежуточного PC2 (пункт 5 на рисунке), который должен иметь установленный шлюз SoMachine, к целевому контроллеру 3 (пункт 4 на рисунке).

1 hop PC 1
2 целевой контроллер 1: 10.128.156.20
3 целевой контроллер 2: 10.128.156.20
4 целевой контроллер 3: 10.128.156.20
5 hop PC 2
6 PC / HMI
7 маршрутизатор – роутер (router)
8 Ethernet

Режим соединения → Nodename через MODEM

Если Вы выбираете опцию **Nodename via MODEM** из списка **Connection Mode**, Вы можете определить контроллер, который размещён позже модемной линии.

1 PC / HMI
2 PC / HMI / MODEM
3 целевой модем
4 целевое устройство
5 телефонная линия

Чтобы установить соединение по модему, нажмите кнопку **MODEM** → **Connect**. В диалоговом окне **Modem Configuration** введите **Phone number** (Номер телефона) целевого модема и сконфигурируйте коммуникационные параметры настройки. Нажмите **OK**, чтобы подтвердить и установить соединение с модемом.

Если шлюз SoMachine остановлен и перезапущен, любое соединение локального шлюза прерывается. SoMachine выводит на экран сообщение, которое должно быть подтверждено, прежде чем будет начат процесс перезапуска.

После того, как соединение с модемом было успешно установлено, кнопка **MODEM** изменяется от **Connect** (Подключение) на **Disconnect** (Разъединение). Список контроллеров очищен и обновлён сканированием модемного соединения для подключенных контроллеров. Вы можете дважды щелкнуть по элементу из списка контроллеров или ввести nodename в текстовое поле **Target Nodename:**, чтобы соединиться с определенным контроллером.

Нажмите кнопку **MODEM** → **Disconnect**, чтобы завершить модемное соединение и остановить и перезапустить шлюз SoMachine. Список контроллеров очищен и обновлён сканированием сети Ethernet.

Коммуникационные параметры настройки

Обзор

Вид **Communication Settings** редактора устройства служит, чтобы сконфигурировать параметры коммуникации между устройством и системой программирования.

Он видим, только если коммуникация между устройством и системой программирования устанавливается при помощи активного пути. Это – настройка по умолчанию для SoMachine V3.1 и более ранних версий. Вы можете выбрать между установлением коммуникации через активный путь и IP-адресом в диалоговое окно **Project Settings** → **Communication settings**. Если выбрана опция **Dial up via “IP-address”**, вид **Controller selection** отображается в редакторе устройства вместо вида **Communication settings**. Это - настройка по умолчанию для SoMachine V4.0 и более поздних версий.

Вид **Communication Settings** редактора устройства

Этот вид разделён на 2 части:

- Левая часть показывает сконфигурированные в настоящее время каналы шлюза в древовидной структуре.
- Правая часть показывает соответствующие данные и информацию.

Описание древовидной структуры

Когда Вы создаете первый проект в своей локальной системе, локальный **Gateway** уже доступен как узел в дереве. Этот шлюз запущен автоматически во время старта системы.

Параметры настройки этого шлюза выведены на экран в правой части окна:

Пример:
Node Name: Gateway-1
Port: 1217
IP-Address: 127.0.0.1
Driver: TCP/IP

Когда шлюз работает, зеленый маркер показан перед узлом **Gateway**; иначе отображается красный маркер. Маркер серый, если со шлюзом еще не было контакта (в зависимости от некоторых протоколов связи, не позволено опросить шлюз, таким образом, состояние не может быть выведено на экран).

Расположенные с отступом ниже узла **Gateway** (открыт/закрыт через знак +/-), Вы будете видеть записи для всех устройств, которые достижимы через этот шлюз. Записям устройства предшествует символ . Записи с целевым ID, отличающимся от устройства, сконфигурированного в настоящее время в проекте, выведены на экран серым шрифтом. Чтобы получить актуальный список доступных в настоящее время устройств, используйте кнопку **Scan network** (скан сети).

Узлы устройств состоят из символа, сопровождаемого именем узла и адресом узла. В правой части окна показываются соответствующие **Target ID**, **Target Name**, **Target Type**, **Target Vendor** и **Target Version**.
В поле **Select the network path to the controller (**выборе сетевого пути к контроллеру) канал шлюза специфицируется автоматически выбором канала в древовидной структуре.

Функция фильтра и сортировки

Вы можете фильтровать и сортировать шлюз и узлы устройства, отображаемые в дереве, с помощью выбора в прямоугольниках в правой части вида:

- **Filter** (Фильтр): Позволяет Вам уменьшать число записей в древовидной структуре до тех устройств, **Target ID** которых соответствует тому устройству, которое в настоящее время конфигурируется в проекте.
- **Sorting order** (Порядок сортировки): Позволяет Вам сортировать записи древовидной структуры в соответствии с **Name** или **Node Address** в алфавитном порядке или порядке возрастания.

Описание кнопок / команд

Для изменения коммуникационной конфигурации доступны следующие кнопки или команды в контекстном меню:

Кнопка / Команда	Описание
Set active path	Эти наборы команд в настоящее время - выбранный канал передачи как активный путь к контроллеру. См. описание команды **Set Active Path**. Двойной щелчок по узлу в древовидной структуре имеет тот же эффект.
Add gateway...	Эта команда открывает диалоговое окно **Gateway**, где Вы можете определить шлюз, который будет добавлен к текущей конфигурации. См. описание команды **Add Gateway**.
Add device...	Эта команда открывает диалоговое окно Add Device, где Вы можете вручную определить устройство, которое будет добавлено к выбранной в настоящее время записи шлюза (Рассмотрите функциональность **Scan network**). См. описание команды **Add Device...**
Edit Gateway...	Эта команда открывает диалоговое окно **Gateway** для редактирования параметров настройки в настоящее время выбранного шлюза. См. описание команды **Edit Gateway...**
Delete selected Device	Эта команда удаляет выбранное устройство из дерева конфигурации. См. описание команды **Delete selected Device**.
Scan for device by address	Эта команда сканирует сеть для устройств, которые имеют адрес, специфицированный здесь в дереве конфигурации. Те, которые найдены, будут представлены в шлюзе с указанным адресом узла, дополненным их именем. Сканирование относится к устройствам ниже того шлюза, в дереве которого в настоящее время выбрана запись. См. описание команды **Scan for Device by Address.**
Scan for device by name	Эта команда сканирует сеть для устройств, которые имеют имена, специфицированные здесь в дереве конфигурации (поиск с учетом регистра). Те, которые найдены, будут представлены в шлюзе с указанным именем, дополненные их уникальным адресом узла. Сканирование относится к устройствам ниже того шлюза, в дереве которого в настоящее время выбрана запись. См. описание команды **Scan for Device by Name**.
Scan for device by IP address	Эта команда сканирует сеть для устройств, которые имеют IP-адрес, специфицированный здесь в дереве конфигурации. Те, которые найдены, будут тогда представлены в шлюзе с указанным адресом узла, дополненным их именем. Сканирование относится к устройствам ниже того шлюза, в дереве которого в настоящее время выбрана запись. См. описание команды **Scan for Device by IP Address.**
Connect to local Gateway	Эта команда открывает диалоговое окно для конфигурации локального шлюза и поэтому обеспечивает альтернативу ручного редактирования файла *Gateway.cfg*. См. описание команды **Connect to local Gateway...**.
Scan network	Эта команда запускает поиск доступных устройств в Вашей локальной сети. Дерево конфигурации связанного с этим шлюза будет соответственно обновлено. См. описание команды **Scan Network**.

Описание опций

Доступны 2 варианта доступной ниже древовидной структуры:

Опция	Описание
Don't save network path in project (Не сохранение сетевого пути в проекте)	Активируйте эту опцию, если текущее определение сетевого пути должно быть сохранено не в проекте, но в локальных параметрах настройки на Вашем компьютере. Поэтому урегулирование пути восстанавливается, если проект вновь открыт на том же самом компьютере. Будет необходимо переопределение, если проект используется в другой системе.
Secure online mode (Безопасный онлайн режим)	Активируйте этот выбор, если из соображений безопасности пользователь должен быть побужден к подтверждению, выбирая одну из следующих онлайн команд: **Force values, Multiple download, Release force list, Single cycle, Start, Stop, Write values**..

Конфигурация

Обзор

Вид **Configuration** доступен только в редакторе устройства, если опция **Show generic device configuration views** в диалоговом окне **Tools** → **Options** → **Device editor** активирована. Вид **Configuration** показывает определенные для устройства параметры, и, если позволено описанием устройства, обеспечивает возможность отредактировать значения параметров.

Вид **Configuration** редактора устройства

DI

I/O Mapping | I/O Configuration

Parameter	Type	Value	Default Value	Unit	Description
Inputs					
I0					
Filter	Enumeration of BYTE	1.5	No	ms	Filtering value
Latch	Enumeration of BYTE	No	No		Latching allow
Event	Enumeration of BYTE	No	No		Event detectio
Bounce Filter	Enumeration of BYTE	0.004	0.004	ms	Filtering value
Run/Stop	Enumeration of BYTE	No	No		Run/Stop inpu
I1					
Filter	Enumeration of BYTE	No	No	ms	Filtering value
Latch	Enumeration of BYTE	No	No		Latching allow
Event	Enumeration of BYTE	No	No		Event detectio
Bounce Filter	Enumeration of BYTE	0.004	0.004	ms	Filtering value
Run/Stop	Enumeration of BYTE	No	No		Run/Stop inpu
I2					
Filter	Enumeration of BYTE	No	No	ms	Filtering value

Вид содержит следующие элементы:

Элемент	Описание
Parameter (Параметр)	имя параметра, не редактируемое
Type (Тип)	тип данных параметра, не редактируемый
Value (Значение)	Прежде всего, значение параметра по умолчанию показано непосредственно или символическим именем. Если параметр может быть изменен (это зависит от описания устройства, нередактируемые параметры показаны серым цветом), щелкните по ячейке таблицы, чтобы открыть и отредактировать структуру или список выбора, чтобы изменить значение. Если значение - спецификация файла, стандартное диалоговое окно для открытия файла открывается двойным щелчком по клетке. Это позволяет Вам выбрать другой файл.
Default Value	значение параметра по умолчанию, не редактируемое
Unit	единица значения параметра (например: ms для миллисекунд), нередактируемый
Description	краткое описание параметра, не редактируемое

Приложения

Обзор

Вид **Applications** редактора устройства обслуживает сканирования и удаления приложений в контроллере. Может быть доступной информация о содержании приложения, а также некоторые детали о свойствах приложения.

Вид **Applications** редактора устройства

Описание элементов

Вид **Applications** обеспечивает следующие элементы:

Элемент	Описание
Applications on the PLC	Это текстовое окно перечисляет имена приложений, которые были найдены в контроллере во время последнего скана (при нажатии на **Refresh**). Если скан еще не был выполнен или если скан не возможен, потому что никакой шлюз не сконфигурирован (*см. стр. 111*) для соединения, отображается соответствующее сообщение.
Remove **Remove All**	Нажмите эти кнопки, чтобы удалить приложение, в настоящее время выбранное в списке, или все приложения из контроллера.
Details	Нажмите эту кнопку, чтобы открыть диалоговое окно, показывая информацию, как определено во вкладке **Information** диалогового окна **Properties** объекта приложения.
Content	Если во **View** → **Properties** → **Application build options** опция **Download Application Info** активирована для объекта применения *(см. SoMachine, Menu Commands, Online Help)*, то дополнительная информация о содержании приложения загружается в контроллер. Нажмите кнопку **Content**, чтобы рассмотреть различные POU в виде сравнения. После нескольких загрузок эта информация уже позволяет Вам сравнивать код нового приложения с доступным в контроллере. Это предоставляет более раздробленную информацию для решений о том, как авторизовываться. Для получения дополнительной информации обратитесь к описанию команды **Login**.
Refresh List	Нажмите эту кнопку, чтобы сканировать контроллер на приложения. Список будет соответственно обновлен

Проверки перед загрузкой приложения в контроллер

Следующие проверки выполняются прежде, чем приложение загружается в контроллер:

- Список приложений на контроллере сравнивается с доступными в проекте. Если обнаружены несоответствия, соответствующие диалоговые окна показаны или для загрузки приложений, еще не доступных на контроллере, или для удаления других приложений из контроллера.
- POU, реализованные внешне в приложении, которое будет загружено, проверены относительно того, доступны ли они также в контроллере. Если они не будут доступны на контроллере, соответствующее сообщение (**unresolved reference(s)** - нерешенная ссылка (ссылки) - будет произведено в окне сообщения, а также в виде **Messages**, если выбрана опция **Download**.
- Параметры (переменные) POU приложения, которое будет загружено, сравниваются с одноименными параметрами POU приложения, уже доступного в контроллере (проверка подписей). В случае, если обнаружены любые несоответствия, соответствующее сообщение (signature mismatch(es) - несоответствие подписи) будет произведено в окне сообщения, а также в виде Messages, если выбрана опция **Download**.
- Если во **View** → **Properties** → **Application build options** активирована опция **Download Application Info**, дополнительная информация о содержании приложения будет загружена в контроллер. См. описание кнопки Content в предыдущей таблице.

Файлы

Обзор

Вид **Files** редактора устройства служит для передачи файлов между хостом и контроллером. Это означает, что Вы можете выбрать любой файл из директории локальной сети, чтобы скопировать его в директорию файлов соединённой в настоящее время системы во время выполнения, или наоборот.

Вид **Files** редактора устройства

Это вид разделён на 2 части:

- Левая часть показывает файлы на **Host** (хосте).
- Правая часть показывает файлы на **Runtime** системе реального времени.

Описание элементов

Вид **Files** снабжен следующими элементами:

Элемент	Описание
	Обновляет список **Runtime**.
	Создает новую папку, в которой Вы можете скопировать файлы.
	Удаляет отобранные файлы или папки из списка.
Location Размещение	Определяет папку соответствующей файловой системы, которая будет использоваться для передачи файлов. Выбирает запись из списка или просматривает в дереве файловой системы.
<< >>	Выбирает файлы, которые будут скопированы в дерево файловой системы. Вы можете выбрать несколько файлов одновременно, или Вы можете выбрать папку, чтобы скопировать все файлы, которые она содержит. Чтобы скопировать отобранные файлы из **Host** в Runtime директорию, щелкните **>>**. Чтобы скопировать отобранные файлы из **Runtime** в директорию **Host**, щелкните **>>**. Если файл еще не доступен в целевой директории, он будет там создан. Если файл с именем уже доступен и не будет защищен от записи, он будет переписан. Если он будет защищен от записи, будет произведено соответствующее сообщение.

Регистрация

Обзор

Вид **Log** (регистрации) редактора устройства служит для отображения событий, которые были зарегистрированы в системе во время выполнения контроллера.

Это касается:

- события при старте или выключении системы (загруженные компоненты и их версии)
- загрузка приложения и загрузка проекта
- ориентированные на клиента записи
- записи в журнале драйверов входов/выходов
- записи в журнале сервера данных

Вид **Log** редактора устройства

Описание элементов

Вид **Log** обеспечивается следующими элементами:

Элемент	Описание
Severity (тяжесть аварийного нарушения)	Случаи регистрации сгруппированы в 4 категориях: ● **warning** (предупреждение) ● **error** (ошибка) ● **exception** (исключение) ● **information** (информация) Кнопки в панели выше списка показывают текущее количество регистраций в соответствующей категории. Нажмите кнопки, чтобы включить или отключить показ записей каждой категории.
Time Stamp (Дата и время регистрации)	дата и время регистрации например, **12.01.2007 09:48**
Description (Описание)	описание события, например, **Import function failed of <CmpFileTransfer>** (функция импорта потерпела неудачу)
Component (Компонент)	Здесь Вы можете выбрать конкретный компонент, чтобы получить только показанные записи в журнале относительно этого компонента. Настройка по умолчанию - **<All components>** (все компоненты).
Logger (Регистратор)	Список выбора обеспечивает доступные регистрации. Настройка по умолчанию - **<Default Logger>**,который определен системой выполнения.
	Обновление списка
	Экспортирует список в файл XML. Стандартное диалоговое окно открывается для того, чтобы сохранить файл. Фильтр файла установлен на **xml-files (*.xml)**. Файл системного журнала снабжен указанным именем файла с расширением .XML в выбранной директории.
	Отображение записей, сохранённых в XML файле, который, возможно, был экспортирован, как описано выше. Открывается стандартное диалоговое окно для выбора файла. Фильтр установлен на **xml-files (*.xml)**). Выберите желаемый файл системного журнала. Записи этого файла будут показаны в отдельном окне.
	Очищает текущую таблицу записей, то есть удаляет все показанные записи.
Offline-Logging	Эта опция не используется в SoMachine.
UTC Time	Активируйте эту опцию, чтобы показать отметку времени системы выполнения. Если опция дезактивирована, показывается отметка локального времени компьютера (согласно часовому поясу операционной системы).

Параметры настройки ПЛК

Обзор

Вид параметров настройки **PLC settings** редактора устройства обслуживает конфигурирование некоторых общих параметров настройки для контроллера.

Вид **PLC settings** редактора устройства

Описание элементов

Вид **PLC settings** обеспечивают следующие элементы:

Элемент	Описание
Application for I/O handling (Приложение для обработки Входов/Выходов):	Определите здесь приложение, назначенное для устройства в **Devices tree**, которое будет управлять обработкой Входов/Выходов. Для SoMachine доступно только одно приложение.
PLC settings (Область параметров настройки ПЛК)	
Update IO while in stop (Обновление Вх/Вых во время остановки)	Если эта опция активирована (по умолчанию), значения каналов входов и выходов обновляются и при остановленном контроллере. В случае истечения времени сторожевого таймера выходы устанавливаются в определенные по умолчанию значения.
Behaviour for outputs in Stop (Поведение выходов во время остановки)	Из списка выбора выберите одну из следующих опций для определения того, как значения в каналах выходов обрабатываются в случае остановки контроллера: ● **Keep current values** (Сохранение текущих значений) Текущие значения не будут изменены. ● **Set all outputs to default** (Установка всех выходов по умолчанию) Будут назначены значения по умолчанию, полученные из разметки. ● **Execute program** (Выполнение программы) Вы можете определить поведение выходов программой, доступной в проекте. Введите здесь имя этой программы, и она будет выполнена при остановке контроллера. Нажмите кнопку **...**, чтобы использовать с этой целью **Input Assistant**.

Элемент	Описание
Update all variables in all devices (Обновление всех переменных во всех устройствах)	Если эта опция активирована, то для устройств текущей конфигурации контроллера, переменные Вх/Вых будут обновляться в каждом цикле задачи цикла шины. Это соответствует опции **Always update variables**. Вы можете установить её отдельно для каждого устройства в виде **I/O Mapping** (*см. стр. 138*).
Bus cycle options (Область опций цикла шины)	
Bus cycle task (Задача цикла шины)	Список выбора предлагает задачи, в настоящее время определенные в **Task Configuration** активного приложения (например, **MAST**). Настройка **MAST** по умолчанию вводится автоматически. **<unspecified** – не специфицирована**>** означает, что задача выбирается согласно внутренним параметрам настройки контроллера, которые поэтому являются зависимыми от контроллера. Она может быть задачей с самым коротким временем цикла, но также может случиться, что с самым долгим временем цикла.

ПРИМЕЧАНИЕ: Установка задачи цикла шины на **<unspecified>** может вызвать непредвиденное поведение Вашего приложения. Проконсультируйтесь с Руководством по программированию, определенным для Вашего контроллера.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННАЯ РАБОТА ОБОРУДОВАНИЯ

Если Вы не уверены в параметрах настройки задачи цикла шины контроллера, не устанавливайте **Bus cycle task** на **<unspecified>**, но выберите специальную задачу из списка.

Отказ следовать этим инструкциям может привести к повреждению оборудования или ране.

Дополнительные уставки

Установка **Generate force variables for IO mapping:** доступна, только если поддерживается устройством. Если опция будет активирована для каждого канала Вх/Вых, который назначен переменной, в диалоговом окне **I/O Mapping** будут создаваться 2 глобальные переменные, как только приложение создано. Эти переменные могут использоваться в визуализации HMI для форсированной (принудительной) установки значения Вх/Вых. Для получения дополнительной информации обратитесь к главе *I/O Mapping* (*см. стр. 135*).

Пользователи и группы

Обзор

Управление пользователями и группами прав доступа отличается в зависимости от используемого Вами контроллера. Для большинства устройств, поддерживающих онлайновое управление пользователями (таких как M258, M241, M251 и LMC ••8), вид **Users and Groups**, описанный в этой главе, используется для управления учетными записями пользователей и пользовательскими группами прав доступа, и связанными правами доступа. Он позволяет Вам управлять доступом в проектах SoMachine и устройствах в режиме онлайн.

Для управления правами пользователей Вы должны войти в систему как **Administrator**.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕИДЕНТИФИЦИРОВАННЫЙ, НЕАВТОРИЗОВАННЫЙ ДОСТУП

- Как можно реже подвергайте контроллеры и сети контроллера воздействию общедоступных сетей и Интернета.
- Используйте дополнительные уровни безопасности, такие как VPN для удаленного доступа, и установите механизмы брандмауэра.
- Ограничьте доступ неавторизованным людям.
- Измените пароли по умолчанию при запуске и часто изменяйте их.
- Проверяйте эффективность этих мер регулярно и часто.

Отказ следовать этим инструкциям может привести к ране или повреждению оборудования.

ПРИМЕЧАНИЕ: Функция **Users and Groups** не предназначена для защиты проекта SoMachine от злонамеренного доступа, а скорее для помощи в предотвращении ошибки пользователей, которым доверяют. Если Вы хотите защитить весь свой проект, активируйте опцию **Enable project file encryption** в диалоговом окне **Project Settings** → **Security** *(см. SoMachine, Menu Commands, Online Help)*. Если Вы хотите защитить только часть своего кода в проекте, поместите этот код в скомпилированной библиотеке (см. *SoMachine, Menu Commands, Online Help*).

ПРИМЕЧАНИЕ: Вы можете использовать связанные с безопасностью команды *(см. SoMachine, Menu Commands, Online Help)*, которые обеспечивают способ добавить, отредактировать и удалить пользователя в онлайн управлении пользователями целевого устройства, куда Вы в настоящее время вошли.

ПРИМЕЧАНИЕ: Вы должны установить пользовательские права доступа, используя программное обеспечение SoMachine. Если Вы клонировали приложение с одного контроллера на другой, Вы должны будете разрешить и установить пользовательские права доступа в предназначенном контроллере.

ПРИМЕЧАНИЕ: Единственный способ получить доступ к контроллеру, который имеет пользовательские права доступа и для которого у Вас нет пароля (паролей) - это произвести операцию **Update firmware**, используя SD-карту, или USB ключ памяти *(обращайтесь к SoMachine Controller Assistant User Guide)*, в зависимости от поддержки Вашего конкретного контроллера, или выполнением скрипта *(см. стр. 821)*. Это эффективно удалит существующее приложение из памяти контроллера, но восстановит способность получения доступа к контроллеру.

Для контроллеров **Soft PLC** специфический вид **Users and Groups (**Пользователи и Группы) и отдельный вид **Access Rights (**Права доступа) отображаются в редакторе устройств. Эти специфические виды описаны в главе *Управление пользователями для Soft PLC* (*см. стр. 869*).

Если Вы хотите, чтобы определенные функции контроллера могли быть выполнены только зарегистрированными пользователями, вид **Users and Groups** позволяет Вам определять пользователей, назначать права доступа и требовать пользовательской идентификации при входе.

Чтобы выполнить эти действия, Вы можете создать пользователей и формировать их права доступа к данным, техническим инструментам и файлам при помощи кнопок в области **User Actions**. Вы можете создать пользовательские группы прав доступа и формировать каждое разрешение индивидуально при помощи кнопок в области **Groups Actions**.

Вид пользователей и групп

Вид **Users and Groups** позволяет Вам управлять доступом пользователей на проектах и устройствах. Пользователи и соответствующие права действительны для всех устройств проекта.

Вид **Users and Groups** редактора устройств:

ПРИМЕЧАНИЕ: Когда Вы применяете конфигурацию прав пользователя к цели, новая конфигурация может быть не принята в расчет для связей, которые уже открыты. Чтобы помочь удостовериться, что конфигурация применена, закройте связи с этой целью или её перезагрузкой, или разъединением кабелей (Ethernet и последовательной линии) по крайней мере на 1 минуту.

Область Access protection

Область **Access protection** вида **Users and Groups** содержит флажок **Enable user management**. Этот флажок по умолчанию не установлен, что означает - управление пользователями не активно. Для того чтобы быть способным управлять аккаунтами пользователей, правами доступа пользовательских групп и назначением прав доступа, активируйте флажок **Enable user management**. Чтобы изменить уставки прав пользователя Вы должны или войти как **Administrator**, или иметь права пользователя **Administrate** для своего логина. Во время своего первого входа как администратора система попросит Вас изменить пароль по умолчанию.

Ниже флажка **Enable user management** присутствует список определённых пользователей (в колонке **User**) и групп прав доступа, присвоенных им (в колонках **Access Protection**, разделённых по типу).

Этот список содержит пять пользователей по умолчанию. Вы имеете ограниченный доступ к этим пользователям по умолчанию, как перечислено в таблице:

Default user Пользователь по умолчанию	**Описание**	**Может быть удален**	**Имя может быть изменено**	**Разрешения могут изменяться**	**Пароль по умолчанию (может изменяться)**
Administrator (Администратор)	Пользователь **Administrator** не имеет ограничений; он используется для конфигурации прав пользователей used.	Нет	Нет	Нет	`Administrator` Должен быть изменен при первом входе.
Everyone (Каждый)	**Everyone** используется, когда Вы входите без логина. Если Вы заблокируете разрешения, Вы должны будете входить как пользователь, у которого есть разрешения.	Нет	Нет	Да	Нет назначенного пароля (password)
USER (Пользователь)	Пользователь **USER** используется Веб-сервером для доступа к контроллеру.	Нет	Да	Да	`USER`
HMI (Панель оператора)	Пользователь **HMI** используется HMI для подключения к контроллеру.	Нет	Да	Да	`HMI`
Developer (Разработчик)	Для пользователя **Developer** есть предоставленные подходящие разрешения для разработки проекта.	Да	Да	Да	`Developer`

Права доступа пользователей содержатся в группах и разделены на три зоны или типы доступа под **Access Protection**:

- **Data Access:** включают доступ к приложениям, предполагающий чтение/запись данных в(из) устройства, такого как HMI, OPC.
- **Engineering Tools:** включают доступ к инструментам программирования, таким как SoMachine and Controller Assistant.
- **File Access:** включают доступ к внутренней или внешней файловой системе, таким как внешний носитель, SoMachine protocol, FTP, и Web.

Нажатие на поле раскрывающегося списка для любого определённого пользователя в любой колонке типа доступа предоставляет список групп прав пользователей для определённого пользователя, из которого Вы можете выбирать для конкретного пользователя и типа доступа. За дополнительной информацией о группах прав доступа обращайтесь к Управлению группами прав доступа *(см. стр. 128)*.

Область действий пользователей

Кнопки доступны, только если активирована опция **Enable user management.**

Нажатие кнопок открывает диалоговое окно, которое вызывает Ваш вход как **Administrator**, потому что только этот пользователь наделён правами управления пользователями.

Кнопки **User Actions** для выполнения стандартных функций управления пользователями:

Кнопка	Описание
Add user...	Нажмите эту кнопку для добавления нового пользователя в список. Открывается диалоговое окно **Add user.** Введите **User Name**, полное **Name**, **Description** и **Password**. Повторите пароль в поле **Confirm password**. Чтобы сделать нового пользователя доступным для использования, разрешите опцию **Activated**.
Delete user...	Нажмите эту кнопку для удаления пользователя, выбранного в списке области **Access protection**
Edit user...	Нажмите эту кнопку для изменения пользователя, выбранного в списке области **Access protection**. Открывается диалоговое окно **Edit user**. Оно соответствует диалоговому окну **Add user** (см. выше), содержащиму установки для текущего определённого пользователя. Чтобы сделать пользователя доступным для использования, разрешите опцию **Activated**.

Измените пароли по умолчанию для всех пользователей по умолчанию (**USER**, **HMI**, **Developer**). В дополнение рассмотрите тщательно последствия для предоставления любых прав доступа пользователю по умолчанию **Everyone**. С одной стороны, предоставление прав доступа пользователю **Everyone** разрешит Вам выполнять скрипты или из USB порта или SD порта (в зависимости от типа Вашего контроллера). С другой стороны, это позволит всем права доступа, предоставленные Вам, без первого входа в систему.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕАВТОРИЗОВАННЫЙ ДОСТУП К ДАННЫМ

- Оперативно изменяйте любые и все пароли по умолчанию на новые, надёжные пароли.
- Не распространяйте пароли неавторизованному или другому неправомочному персоналу
- Ограничьте права доступа пользователю **Everyone** только необходимыми для Вашего приложения.

Отказ следовать этим инструкциям может привести к смерти, серьёзным травмам или повреждению оборудования.

ПРИМЕЧАНИЕ: Надёжный пароль – это такой, который не был распространён какому-то неправомочному персоналу и не содержит какой-то персональной или другой очевидной информации. Кроме того, смесь заглавных и строчных букв и цифр повышает уровень надёжности. Вы должны использовать пароль длиной не менее 7 символов.

Область групповых действий

Кнопки доступны, только если активирована опция **Enable user management**.

Нажатие кнопок открывает диалоговое окно, которое вызывает Ваш вход как **Administrator**, потому что только этот пользователь наделён правами управления пользователями.

Кнопки, используемые для выполнения функций управления правами доступа по группам.

Кнопка	Описание
Add group	Нажатие этой кнопки создаёт новую группу. Открывается диалоговое окно **Custom groups**.
Edit group	Нажатие этой кнопки изменяет группу. Открывается диалоговое окно **Custom groups**. Из списка **Group to edit** выберите группу, которую Вы хотите редактировать. Уставки, определённые для выбранной группы, отображаются ниже в таблице.
Delete group...	Нажатие этой кнопки удаляет группу, выбранную в списке области **Access protection**. Примечание: Некоторые группы по умолчанию не могут быть удалены *(см. стр. 123)*.

Группы прав доступа для типов защиты доступа

По умолчанию число групп прав доступа предопределено и доступно из раскрывающегося списка под индивидуальным типом **Access Protection** для каждого определённого пользователя.

Таблица перечисляет имеющиеся доступные по умолчанию группы прав доступа по типу **Access Protection**:

Тип Access Protection	Права доступа по умолчанию	Описание
Data Access	**None**	Доступ закрыт.
	Read	● Доступ к Веб-сайту устройства. ● Чтение незащищённых данных.
	Read-Write	● Доступ к Веб-сайту устройства. ● Чтение/запись незащищённых данных.
Engineering Tools	**None**	Доступ закрыт.
	Monitor	● Логин. ● Выгрузка исходной программы ● Переменные только для чтения
	Operate	● Логин. ● Выгрузка исходной программы ● Форсирование/запись переменных
	Program	Нет ограничений, исключая разрешение конфигурации прав пользователей.
	Administrate	Нет ограничений, даже разрешена конфигурация прав пользователей.
File Access	**None**	Доступ закрыт.
	Read	Чтение незащищённых файлов.
	Read-Write	Чтение/запись незащищённых файлов
	Read-Write-Delete	Чтение/запись/удаление незащищённых файлов.

Управление группами прав доступа

В дополнение к группам с правами доступа по умолчанию Вы можете создать свою собственную группу пользователей. Вы можете конфигурировать права доступа для специфических инструментов или команд, объединённых под каждым типом **Access Protection** (**Data Access**, **Engineering Tools**, **File Access**). Для получения этого нажмите кнопку **Add group** или **Edit group** из зоны **Groups Actions**. Открывается диалоговое окно **Custom groups**.

Параметр	Описание
Name	Введите **Name** для группы, используя максимум 32 символа. Разрешены символы a–z, A–Z, 0–9 символ нижней черты.
Access Type	Выберите одну из возможных **Access Type** на группу: ● **Data Access** ● **Engineering Tools** ● **File Access** **Actions** и **Permissions**, доступные для выбранного **Access Type**, перечислены ниже с левой стороны.

Параметр	Описание
Reset to	Для копирования прав доступа пользователей от одной группы к другой выберите группу из списка **Reset to**. Это переустановит значения **Actions** и **Permissions** выбранной Вами группы.
Список **Actions** / **Permissions**	Этот список показывает **Actions** и **Permissions**, доступные для выбранного **Access Type**. Обращайтесь к спискам действий и разрешений по типу доступа, приведенным ниже.
Список **TargetName** / **Permissions**	**TargetName** – это список точно того, что Вы искали в дереве **Devices tree** Вашего проекта для выбранного вами устройства. Вы можете конфигурировать разрешения для каждого устройства индивидуально. Для достижения этого выберите устройство в списке и выберите подходящую опцию (**Default**, **Deny**, **Read only**, **Modifiable**, **Full**) из списка **Permissions**. Значение **Default** соответствует разрешениям для пользователя **Everyone**.

Таблица перечисляет доступные **Actions** (действия) и **Permissions** (разрешения), которые Вы можете назначить для **Access Type** → **Data Access**:

Доступные действия	Разрешения
OPC Server Позволяет пользователю соединяться с устройством с OPC сервером	• **Deny** • **Read** • **Read - Write**
HMI Позволяет пользователю доступ к Vijeo-Designer, используя параметры логин / пароль.	• **Deny** • **Read** • **Read - Write**
Web Server Позволяет пользователю доступ к Веб-серверу, используя параметры логин / пароль.	• **Deny** Пользователь не может иметь доступа к Веб-серверу. • **Read** Пользователь может иметь доступ к Веб-серверу и читать переменные. • **Read - Write** Пользователь может иметь доступ к Веб-серверу и читать и записывать переменные.

Таблица перечисляет доступные **Actions** (действия) и **Permissions** (разрешения), которые Вы можете назначить для **Access Type** → **Engineering Tools**:

Доступные действия	Разрешения
Login Позволяет пользователю соединяться с устройством и получать доступ к приложению. Это разрешение является предпосылкой для назначения любого другого онлайн разрешения.	• **Default** • **Deny** • **Grant**
Reset	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** Пользователь не может выполнить перезапуск устройства. • **Cold** Пользователь может выполнить холодный перезапуск устройства. • **Cold - Warm** Пользователь может выполнить холодный и тёплый перезапуск устройства.
Run / Stop	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** Пользователь не может выполнить запуск/остановку приложения. • **Grant** Пользователь не может выполнить запуск/остановку приложения.
Logger Позволяет пользователю доступ к виду **Log** редактора устройства.	• **Default** • **Deny** • **Grant**
Variables	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** Пользователь может читать/записывать любые переменные. • **Read** Пользователь может читать переменные приложения. • **Read - Write** Пользователь может читать, записывать и форсировать незащищённые переменные.
Trace Позволяет пользователю доступ к устройству трассировки.	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** • **Grant**

Доступные действия	Разрешения
Project Settings Позволяет пользователю изменять (читать и записывать) информацию и параметры проекта.	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** • **Grant**
PLC Settings Позволяет пользователю доступ к виду **PLC Settings** редактора устройства.	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** • **Grant**
Download Позволяет пользователю загружать все приложения или текущее активное приложение в устройство.	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** • **Grant**
Online Change Позволяет пользователю выполнять команду **Online Change** (онлайн изменения).	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** • **Grant**
Image Download Позволяет пользователю загружать изображения.	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** • **Grant**
Upload Source Позволяет пользователю выполнять команду **Source Upload** (выгрузки исходной программы из устройства).	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** • **Change**
Image Upload Позволяет пользователю выгружать изображения.	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** • **Change**
Update Device Позволяет пользователю выполнять команду **Update Device** (обновление устройства).	• **Default** • **Deny** • **Change**
Project Compilation Позволяет пользователю выполнять команды построения **Build** и **Build all**.	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** • **Change**
Device Management Позволяет пользователю добавлять, изменять, удалять, обновлять, восстанавливать устройство.	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** • **Change**

Доступные действия	Разрешения
Create Object Позволяет пользователю создавать объекты в проекте.	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** • **Change**
Developing Tools Позволяет пользователю изменять приложения, POUs, устройства.	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** • **Change**
Visualization Tools Позволяет пользователю добавлять и редактировать Веб-визуализации и создавать визуализации в текущем проекте.	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** • **Change**
Library Management Позволяет пользователю создавать осуществлять управление библиотеками, исключая список библиотек в проекте и показ их свойств.	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** • **Change**
Recipe Позволяет пользователю создавать, редактировать, посылать, выгружать, выполнять рецепт и загружать данные из текущей программы в рецепт.	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** • **Change**
Debugging Tools Позволяет пользователю выполнять команды отладки, включая точки останова.	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** • **Change**
SoftMotion Tools Позволяет пользователю выполнять команды CAM и CNC.	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** • **Change**
DTM Management Позволяет пользователю выполнять команды FDT (field device tool – инструмента полевого устройства).	• **Default** Некоторые разрешения для пользователя **Everyone**. • **Deny** • **Change**

Таблица перечисляет доступные **Actions** (действия) и **Permissions** (разрешения), которые Вы можете назначить для **Access Type** → **File Access**:

Доступные действия	Разрешения
File system **FTP** **Removable media** Позволяет пользователю доступ к файлу, используя FTP, SoMachine или Веб.	• **Deny** • **Read** • **Read - Write** • **Read - Write - Delete**

Task Deployment – развёртывание задач

Обзор

Вид **Task deployment** редактора устройств показывает таблицу с входами/выходами и их назначением к определённым задачам. Прежде чем информация может быть отображена, должен быть скомпилирован проект и должен быть сгенерирован код. Эта информация помогает в обнаружении ошибок в случае, когда один и то же вход/выход обновляется в разных задачах с различными приоритетами.

Task deployment редактора устройства

Таблица показывает задачи, отсортированные по их приоритетам. Нажмите на заголовке (имени задачи) для отображения только переменных, назначенных на эту задачу. Чтобы показать снова все переменные, нажмите первую колонку (**I/O channels**).

Для открытия таблицы разметки Вх/Вых канала дважды нажмите вход или выход. Синие стрелки указывают на задачу цикла шины.

В показанном выше примере переменная **usiBK5120Out AT %QB0** используется в двух различных задачах. В этой ситуации выход, установленный одной задачей, может быть перезаписан другой задачей: это может привести к неопределённому значению. В общем случае не рекомендуется записывать ссылки выходов в больше, чем одну задачу, так как это может сделать программу трудной для отладки и часто может привести к непреднамеренным результатам в действии Вашей машины или процесса.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННОЕ ДЕЙСТВИЕ ОБОРУДОВАНИЯ

Не записывайте выходную переменную в более чем одной задаче.

Отказ следовать этим инструкциям может привести к смерти, серьёзным травмам или повреждению оборудования.

Статус

Обзор

Вид **Status** редактора устройства показывает статусную информацию (например, **Running**, **Stopped**) и специфические диагностические сообщения от соответствующего устройства; также по используемой карте и внутренней шине системы.

Информация

Обзор

Вид **Information** редактора устройства показывает некоторую общую информацию по устройству, выбранному в данный момент в **Devices tree**: **Name** (имя)**, Vendor** (поставщик), **Type** (тип), **Version number** (номер версии)**, Order Number** (каталожный номер), **Description** (описание), **Image** (изображение).

Раздел 6.2
I/O Mapping – Разметка Вх/Вых

Содержание раздела

Этот раздел содержит следующие темы:

Тема	Страница
Разметка Вх/Вых	136
Работа с диалогом I/O Mapping	138
I/O Mapping в режиме онлайн	141
Неявные переменные для форсированной установки Вх/Вых	142

I/O Mapping - Разметка Вх/Вых

Обзор

Вид **I/O Mapping** редактора устройства именуется как **<devicetype> I/O Mapping** (например, **PROFIBUS DP I/O Mapping**). Он обслуживает конфигурацию разметки Вх/Вых контроллера. Это означает, что переменные проекта, используемые приложением, назначаются на адреса входов, выходов и памяти контроллера.

Определите приложение, которое будет обслуживать Вх/Вых в виде **PLC settings** *(см. стр. 121)*.

ПРИМЕЧАНИЕ: Если это поддерживается устройством, Вы можете использовать режим онлайн конфигурации для доступа к аппаратным средствам Вх/Вых без предварительно загруженного приложения. Для получения дополнительной информации обращайтесь к описанию Online Config Mode *(см. SoMachine, Menu Commands, Online Help)*.

Смотрите следующие разделы:
- Работа с диалогом I/O Mapping *(см. стр. 138)*
- I/O Mapping в онлайн режиме *(см. стр. 141)*
- Неявные переменные для форсированной установки Вх/Вых *(см. стр. 142)*

Общая информация по разметке Вх/Вых на переменные

Может ли разметка Вх/Вых быть сконфигурирована для текущего устройства, зависит от устройства. Может быть, что вид используется только для того, чтобы показать неявно созданный экземпляр устройства. Смотрите описание *IEC объектов (см. стр. 140)*.

Главным образом, помните следующее по разметке Вх/Вых на переменные:
- Переменные, требующие входа, не могут быть доступны по записи.
- Существующая переменная может быть размечена на один вход.
- Вместо использования вида **I/O Mapping**, Вы также можете назначить адрес переменной через AT декларацию *(см. стр. 508)*.

  Однако рассмотрите следующее:
  - Вы можете использовать AT декларации только для локальных или глобальных переменных; однако, не для входных и выходных переменных POUs.
  - Возможность генерации форсируемых переменных для Вх/Вых (обращайтесь к «Неявные переменные для форсированной установки Вх/Вых» *(см. стр. 142)*) не будет доступна для AT деклараций.
  - Если AT декларации используются с членами структур или функциональных блоков, все экземпляры будут иметь доступ к одному и тому же месту памяти. Это место памяти соответствует статическим переменным в классических языках программирования, таких как C.
- Расположение памяти структур определяется целевым устройством.
- Для каждой переменной, которая назначена на канал Вх/Вых в виде **I/O Mapping**, форсируемая переменная может быть создана во время выполнения построения приложения *(см. SoMachine, Menu Commands, Online Help)*. Вы можете использовать их для форсирования значений входов или выходов во время ввода машины в действие, например, через визуализацию (HMI). Обращайтесь к разделу *Неявные переменные для форсированной установки Вх/Вых (см. стр. 142)*.

Автоматическая разметка Вх/Вых

Версии SoMachine V4.0 и более поздние обеспечивают функцию автоматической разметки Вх/Вых. Она автоматически создаёт IEC переменные, как только устройство или модуль с модулями Вх/Вых добавляется в **Devices Tree** и размещает их для каждого входа и/или выхода. По умолчанию функция активирована.

Вы можете деактивировать и сконфигурировать функцию в диалоговом окне **Project** → **Project Settings** → **Automatic I/O mapping**.

Диалоговое окно содержит следующие элементы:

Элемент	Описание
Create variable on I/Os mapping	Выбор или отмена этой опции активирует или деактивирует функцию автоматической разметки Вх/Вых.
Область **Mapping**	
Bitwise	Выберите эту опцию, чтобы создать переменные для каждого бита.
Module-wise	Выберите эту опцию, чтобы создать переменную для каждого модуля, не для отдельных битов.
Область **Naming Rule**	
Текстовое поле	Вводите следующие символы, предваряемые символом `#`, чтобы специфицировать части, из которых будет состоять имя переменной: ● Введите `#X`, чтобы вставить `i` для входов и `q` для выходов в имя переменной. ● Введите `#T`, чтобы вставить префикс код для соответствующего типа данных переменной в имя переменной. Префиксы, которые используются для различных типов данных, перечислены в главе *Рекомендации по наименованию переменных (см. стр. 500)*. ● Введите `#D`, чтобы вставить имя устройства в имя переменной. ● Введите `#C`, чтобы вставить имя, определённое в колонке **Channel** в имя переменной.

Работа с диалогом I/O Mapping

Обзор

Следующее является иллюстрацией вкладки **I/O Mapping** редактора устройств

Описание элементов в области Channels

Вкладка **I/O Mapping** tab обеспечивает следующие элементы в области **Channels**, если это предусмотрено устройством:

Элемент	Описание
Channel	символическое имя канала входов или выходов устройства
Address	адрес канала, например: `%IW0`
Type	тип данных канала входов или выходов, например: BOOL Если тип данных нестандартный, но структура или поле битов определены в описании устройства, они будут перечислены, только если они являются частью IEC стандарта. Это индицируется как IEC тип в описании устройства. В противном случае поле в таблице будет пустым.
Unit	единица значения параметра, например: **ms** для миллисекунд
Description	краткое описание параметра
Current Value	текущее значение параметра, отображаемое в онлайн режиме

Конфигурация I/O Mapping

Произведите разметку Вх/Вых с помощью назначения соответствующих переменных проекта каналам входов и выходов устройства, каждую в колонке **Variable**.

- Тип канала уже индицируется в колонке **Variable** символом: для входа, для выхода. В этой строке введите имя или путь к переменной, на которой канал должен быть размечен. Вы можете или размещать существующую переменную проекта, или определить новую переменную, которая затем будет автоматически декларирована как глобальная переменная.
- При разметке структурированных переменных редактор будет предотвращать, чтобы и переменная структуры (например, `%QB0`) и конкретные элементы структуры (например, в этом случае `%QB0.1` и `QB0.2)` могли быть введены.
  Это означает: Когда есть основное поле ввода с подструктурой битовых полей канала в таблице разметки (как показано на рисунке выше), тогда или в строке главного поля может быть введена переменная, или в подэлементах (битовых каналах), никогда одновременно.
- Для размещения существующей переменной укажите полный путь. Например: <application name>.<pou path>.<variable name>';
  Пример: `app1.plc_prg.ivar`
  Для этой цели может быть полезно открыть помощник ввода, используя кнопку **...**. В колонке **Mapping** символ будет отображаться, и значение адреса будет перечёркнуто. Это не означает, что этот адрес памяти больше не существует. Однако он не используется непосредственно, потому что значение существующей переменной управляется по другому расположению в памяти, и особенно в случае выходов, другая уже существующая переменная не должна сохраняться по этому адресу (`%Qxx` в разметке Вх/Вых), чтобы предотвратить двусмысленности во время записи значений.

Смотрите в следующем примере разметку выхода на существующую переменную `xBool_4`:

- Если Вы хотите определить новую переменную, введите желаемое имя.
  Пример: `bVar1`
  В этом случае символ будет вставлен в колонку **Mapping** и переменная будет внутренне декларирована как глобальная. Отсюда переменная будет доступна глобально в приложении. Диалог разметки является другим местом для декларации глобальных переменных.
  ПРИМЕЧАНИЕ: Альтернативно адрес также может быть прочитан или записан в программном коде, таком как ST (структурированный текст).
- Рассмотрите возможность изменений в конфигурации устройства, рекомендуется делать разметку в диалоге конфигурации устройства.

ПРИМЕЧАНИЕ: Если UNION представлено каналами Вх/Вых в диалоге разметки, от устройства зависит, является ли корневой элемент размещаемым или нет.

Если декларированная переменная данного типа больше чем то, где она размещена, значение отображаемой переменной будет назначено усечённым до размера размеченной целевой переменной.

Например, если переменная декларирована как тип данных WORD и она размечена на BYTE, на байт будут размечены только 8 бит слова.

Подразумевается, что для мониторинга значения в диалоге разметки значение, отображаемое в корневом элементе адреса, будет значением декларированной переменной – как непрерывно проверяется в проекте. В подэлементах ниже корня значения конкретных элементов размеченных переменных будут наблюдаться. Однако только часть декларированного значения может быть отображена среди подэлементов.

Следующее подразумеваемое – когда Вы задаёте декларированные переменные на физические выходы. Также, если Вы задаёте тип данных больше, чем тип данных выхода, он может получить урезанное значение, что может воздействовать на Ваше приложение непредназначенными путями.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННОЕ ДЕЙСТВИЕ ОБОРУДОВАНИЯ

Проверьте, что декларированный тип данных, который размечен на физические Вх/Вых, совместим с предназначенным действием Вашей машины.

Отказ следовать этим инструкциям может привести к смерти, серьёзным травмам или повреждению оборудования.

Элемент	Описание
Reset mapping	Нажмите эту кнопку, чтобы восстановить уставки разметки на значения по умолчанию, определённые в файле описания устройства.
Always update variables	Если эта опция активирована, переменные будут обновляться в каждом цикле задачи цикла шины *(см. стр. 121)*, не имеет значения, используются ли они или они назначены на ввод или вывод.

IEC Objects

Эта часть вкладки доступна, только если безусловно был создан экземпляр объекта устройства, который мог быть доступен приложению (например, с целью перезапустить шину или информацию об опросе) Доступен ли такой экземпляр и как он может быть использован, зависит от контроллера и описывается в его руководстве по программированию.

Bus Cycle Options

Эта опция конфигурации будет доступной для устройств с циклическими вызовами до и после чтения входов и выходов. Она позволяет Вам установить специфическую для устройства задачу цикла шины *(см. стр. 121)*.

По умолчанию будет верной уставка цикла родительской шины (**Use parent bus cycle setting**). Это означает, что **Devices Tree** будет подвергнуто поиску следующего определения верной задачи цикла шины.

Чтобы назначить специфическую задачу цикла шины, выберите желаемую из списка выбора. Список предоставляет задачи, определённые в настоящее время в конфигурации задач приложения.

I/O Mapping в онлайн режиме

I/O Mapping в онлайн режиме

Если структурная переменная размещается на корневом элементе адреса (самом верхнем в дереве соответствующего адреса в диалоге разметки), тогда в онлайн режиме значение не будет отображаться в этой строке. Однако, если, например, переменная DWORD размечена на этот адрес, тогда в корневой строке, так же как линии битовых каналов, размещённой ниже, соответствующие значения будут наблюдаться. Обычно поле корневой строки остаётся пустым, если значение составлено из многочисленных подэлементов.

Неявные переменные для форсированной установки Вх/Вых

Обзор

Во время ввода в действие установки или машины, может быть, необходимо форсированно задать значения Вх/Вых, например, с помощью визуализации. Для этой цели Вы можете генерировать специальные форсирующие переменные для каждого канала Вх/Вых, который которые размечаются на переменную во вкладке **I/O Mapping** редактора устройства.

В качестве предусловия должна быть активирована уставка **Generate force variables for IO mapping** во вкладке **PLC settings**. Тогда при каждом выполнении построения приложения для каждого размеченного канала Вх/Вых будут генерироваться две переменные согласно следующему синтаксису. Любые свободные пробелы в имени канала будут заменены нижней чертой.

<devicename>_<channelname>_<IECaddress>_Force типа BOOL, для активирования и деактивирования форсировки

<devicename>_<channelname>_<ECaddress>_Value типа данных канала, для определения значения, форсированно (принудительно) устанавливаемого на канал

Эти переменные будут доступны в ассистенте ввода в категории конфигурации **Variables** → **IoConfig_Globals_Force_Variables**, и таким образом, внутри системы программирования.

Нарастающий фронт форсирующей переменной активирует форсирование соответствующего Вх/Вых значением, определённым переменной значения. Спадающий фронт деактивирует форсирование. Деактивация установкой форсирующей переменной назад в FALSE необходима, перед тем как новое значение может быть форсировано.

Рассмотрите ограничения, перечисленные ниже.

Пример

Если разметка завершена, как показано на рисунке вкладки **I/O Mapping** редактора устройства *(см. стр. 138)*, тогда при построении (F11) будут сгенерированы следующие переменные приложения и будут доступны в помощнике ввода:

- `Digitax_ST_Control_word_QW0_Force : BOOL;`
- `Digitax_ST_Control_word_QW0_Value : UINT;`
- `Digitax_ST_Target_position_QD1_Force : BOOL;`
- `Digitax_ST_Target_position_QD1_Value : DINT;`
- `Digitax_ST_Status_word_IW0_Force : BOOL;`
- `Digitax_ST_Status_word_IW0_Value : UINT;`
- `Digitax_ST_Position_actual_value_ID1_Force : BOOL;`
- `Digitax_ST_Position_actual_value ID1_Value : DINT;`

Ограничения

- Только каналы, которые размечены на переменную во вкладке **I/O Mapping** (то есть, переменную, определённую в колонке **Variable**, не важно, она новая или существующая) могут быть форсированы с помощью описанных выше неявных переменных.
- Неиспользуемые входы / выходы, а также те, которые размечаются через AT декларацию в программе приложения, не могут быть форсированы.
- Соответственные каналы Вх/Вых должны быть использованы, по крайней мере, в одной задаче.
- Форсированные Вх/Вых не индицируются в мониторинге (вид просмотра watch view, диалог I/O mapping). Значение используется только неявно в драйвере Вх/Вых для записи в устройство.
- Форсированные входы отображаются корректно красным символом форсирования (**F**), однако не форсированными входами/выходами.

Часть III
Программа

Содержание этой части

Эта часть содержит следующие главы:

Глава	Наименование главы	Страница
7	Компоненты программы	147
8	Конфигурация задач	219
9	Управление приложениями	223

Глава 7

Компоненты программы

Что находится в этой главе?

Эта глава содержит следующие разделы:

Раздел	Тема	Страница
7.1	Program Organization Unit (POU) – Единица организации программы	148
7.2	Function Block – Функциональный блок	174
7.3	Application Objects – Объекты приложения	192
7.4	Application - Приложение	218

Раздел 7.1

Program Organization Unit (POU)

Что в этом разделе

Этот раздел содержит следующие темы:

Тема	Страница
POU	149
Добавление и вызов POUs	150
Программа	154
Функция	156
Метод	158
Свойства	161
Интерфейс	163
Свойства интерфейса	167
Действие	170
Внешняя функция, функциональный блок, метод	172
POUs для неявных проверок	173

POU

Обзор

Термин Program Organizational Unit (POU) используется для всех объектов программирования (программ, функциональных блоков, функций, и т.д.), которые используются для создания приложения контроллера.

Управление POU

POUs, которые управляются в узле **Global** дерева **Applications tree**, не являются специфицированными на устройства, но они могут быть иллюстрированы примерами для использования в устройствах (приложение). Для этих целей POUs программы должны быть вызваны задачей соответствующего приложения.

POUs, которые вставлены в **Applications tree** явно ниже приложения, могут быть представлены приложениями, сдвинутыми ниже этого приложения (дочернее приложение). Дальнейшую информацию смотрите в описании **Devices tree** *(см. стр. 38)* и объекта **Application** *(см. стр. 218)*.

Но POU - это также имя определённой подкатегории этих объектов в меню **Add Object**. В этом месте как раз сосредоточены программы, функциональные блоки и функции.

Следовательно, объект POU в общем случае является программной единицей. Это объект, который управляется или в не специфицированным на устройство узле **Global** дерева **Applications tree**, или непосредственно ниже приложения в **Applications tree**. Она может быть просмотрена и отредактирована в виде редактора. Объект POU может быть программой, функцией или функциональным блоком.

Возможно установить определённые **Properties** (свойства – такие как условия построения и т.п.) для каждого конкретного объекта POU.

За описанием того, как создать POU, обращайтесь к секции *Добавление POUs в приложение (см. стр. 151)*. POU, которые Вы создали, добавляются в вид **Assets** каталога **Software catalog**.

Вы можете добавить POU, доступные в виде **Assets**, в проект двумя различными путями:
- Выберите POU в виде **Assets** и перетащите её в подходящий узел в **Applications tree**.
- Выберите POU в виде **Assets** и перетащите её в вид логического редактора *(см. стр. 247)*.

Среди объектов POU есть объекты устройств, используемые для выполнения программ в целевых системах (**Resource**, **Application**, **Task Configuration** и т.п.). Они управляются в **Applications tree**.

Добавление и вызов POUs

Введение

Вы можете добавлять единицы организации программы Program Organization Units (POUs) в Ваше приложение в **Software catalog** → **Assets** или в **Applications tree**.

Различными типами POU являются:

- **Program**: Она возвращают одно или несколько значений во время выполнения. Все значения сохраняются от последнего времени выполнения программы до следующего. Она может быть вызвана другой POU.
- **Function Block**: Он производит одно или более значений во время выполнения программы. В противоположность функции значения выходных переменных и необходимые внутренние переменные будут сохраняться от одного выполнения функционального блока до другого. Таким образом, вызов функционального блока с неизменными аргументами (входными параметрами) приводит не всегда к получению неизменных выходных значений.
- **Function**: Она производит только один элемент данных (который может состоять из нескольких элементов, таких как поля или структуры) когда она выполняется. Вызов в текстовых языках может иметь место как оператор в выражениях.

Добавление POUs в Application

Чтобы добавить POU в приложение контроллера, проделайте следующее:

Шаг	Действие
1	В секции **Software catalog** → **Assets** → **POUs** выберите узел **Application**, нажмите кнопку зелёного плюса и выполните команду **POU...**. В качестве альтернативы Вы можете нажать правой кнопкой мыши узел **Application** контроллера и выбрать **Add Object** → **POU**. Два метода доступны также в **Applications tree**. **Результат:** Открывается диалоговое окно **Add POU.**
2	В диалоговом окне **Add POU** назначьте имя Вашей POU вводом имени в текстовое поле **Name**. **Примечание:** Имя не должно содержать пробелов. Если Вы не введёте имени, имя будет дано по умолчанию. Назначение содержательного имени для POU может облегчить организацию Вашего проекта.

Шаг	Действие
3	Выберите тип POU, который Вы хотите: ● **Program - Программа** ● **Function Block** - Функциональный блок: **a.** Выберите **Extends** и нажмите просмотр, чтобы выбрать функцию блока, которую Вы хотите, в **Input Assistant**. **b.** Нажмите кнопку **OK**. **c.** Выберите **Implements** и нажмите просмотр, чтобы выбрать интерфейс, который Вы хотите, в **Input Assistant**. **d.** Нажмите кнопку **OK**. **e.** В окне списка **Method implementation language** выберите язык программирования, который Вы хотите, для редактирования функционального блока. ● **Function** – Функция: **a.** Нажмите кнопку просмотра, чтобы выбрать **Return type** который Вы хотите, в **Input Assistant**. **b.** Нажмите кнопку **OK**.
4	Из окна списка **Implementation Language**, выберите язык программирования, который Вы хотите, для редактирования Вашей программы.
5	Нажмите кнопку **Open**.

Уже определённые POUs перечисляются в секции **Software catalog** → **Assets** → **POUs**. Вы можете добавлять их в Ваше приложение перетаскиванием их в **Applications tree** и отпусканием их на узле **Application**. Вы можете также опустить POU в виде логического редактора.

Назначение POUs на задачу (Task)

По крайней мере, одна POU должна быть назначена на задачу. Чтобы добавить POU к задаче, проделайте следующее:

Шаг	Действие
1	Под узлом **Task Configuration** контроллера дважды нажмите на задаче, к которой вы хотите добавить Вашу POU. Во вкладке **Configuration** нажмите **Add Call**. Альтернативно в **Applications tree** выберите узел задачи, в котором Вы хотите декларировать Вашу POU и нажмите кнопку зелёного плюса. Выполните команду **POU...** из списка. Нажмите кнопку **...**. **Результат:** Открывается диалоговое окно **Input Assistant**.
2	Во вкладке **Categories** диалогового окна **Input Assistant** выберите **Programs (Project)**.
3	Нажмите на флажок **Structured view**, чтобы его очистить.
4	В панели **Items** выберите POU, которую Вы хотите.
5	Нажмите кнопку **OK**.

Вызов POUs

POUs могут вызывать другие POUs. Однако рекурсия не разрешена (POU, которая вызывает саму себя).

Когда POU, назначенная на приложение, вызывает другую POU только по её имени (без какого-либо добавленного пространства имени (namespace)) *(см. стр. 707)*, рассмотрите следующий порядок просмотра проекта для вызова POU:

1.	Текущее приложение
2.	**Library Manager** в текущем приложении в **Tools tree**
3.	**Global** узел в **Applications tree**
4.	**Library Manager** в **Global** узле в **Tools tree**

Если POU с именем, указанным в вызове, доступна в библиотеке **Library Manager** приложения, так же как объект в узле **Global** дерева **Applications tree**, нет синтаксиса для внешнего вызова POU в узле **Global** дерева **Applications tree**, используется только её имя. В этом случае переместите соответствующую библиотеку из **Library Manager** приложения в **Library Manager** узла **Global** в **Applications tree**. Затем Вы можете вызывать POU из узла **Global** в **Applications tree** просто по её имени (и, если необходимо, - из библиотеки через предварение пространства имени (namespace)).

Также обращайтесь к главе *POUs для неявных проверок (см. стр. 173)*.

Программа

Обзор

Программа – это POU, которая возвращает одно или несколько значений во время работы. Все значения сохраняются с последнего времени выполнения программы до следующего.

Добавление программы

Для добавления программы в существующее приложение выберите узел приложения в **Applications tree**, нажмите кнопку зелёного плюса и выполните команду **POU...**. Как альтернатива, нажмите правой кнопкой узел **Application** и выполните команду **Add Object** → **POU** из контекстного меню. Для добавления независимой от приложения POU, выберите узел **Global** в **Applications tree** и выполните те же команды.

В диалоговом окне **Add POU** выберите опцию **Program**, введите имя для программы и выберите желаемый язык реализации. Нажмите **Open** для подтверждения. Будет открыт вид редактора для новой программы, и Вы сможете начать редактирование программы.

Декларирование программы

Синтаксис:

PROGRAM <program name>

Далее следуют объявления переменных входов *(см. стр. 514)*, выходов *(см. стр. 515)* и переменных программы. Переменные доступа также возможны как опции.

Пример программы

Вызов программы

Программа может быть вызвана другой POU. Однако, вызов программы в функции *(см. стр. 156)* не разрешён. Экземпляров программ нет.

Если POU вызвала программу, и если значения программы были изменены, эти изменения будут сохраняться до получения программой нового вызова. Это применимо, даже если она будет вызвана из другой POU. Примите во внимание, что это отличается от запроса функционального блока. При вызове функционального блока изменяются переменные только в данном экземпляре функционального блока. Изменения влияют, только когда тот же экземпляр вызывается снова.

Чтобы установить входные и/или выходные параметры в ходе вызова программы, в редакторах текстовых языков (например, ST) назначьте значения параметров после имени программы в круглых скобках.

Для входных параметров используйте `:=` для этого назначения, как при инициализации переменных *(см. стр. 505)* в позиции декларации. Для входных параметров используйте `=>`. Смотрите следующий пример.

Если программа вставлена через **Input Assistant** с использованием опции **Insert with arguments** в виде представления редактора текстового языка, это будет отражено автоматически согласно этому синтаксису для всех параметров, хотя Вы не обязательно должны назначать эти параметры.

Пример для вызовов программы

Программа в IL:

```
CAL                 PRGexample              (
         in_var:= 33                        )
LD                 PRGexample.out_var
ST                 erg
```

Пример с назначением параметров (**Input Assistant** использует опцию **Insert with arguments**):

Программа в IL с аргументами

```
CAL                 PRGexample              (
         in_var:= 33                        ,
        out_var=> erg                       )
```

Пример в ST

```
PRGexample(in_var:= 33);
erg := PRGexample.out_var;
```

Пример с назначением параметров (**Input Assistant** использует опцию **Insert with arguments** как описано ранее):

```
PRGexample (in_var:=33, out_var=>erg );
```

Пример в FBD

Программа в FBD

Функция

Обзор

Функция – это POU, которая выдаёт только один элемент данных (который может состоять из нескольких элементов, таких как поля или структуры) когда она выполняется. Вызов в текстовых языках может иметь место как оператор в выражениях.

Добавление функции

Чтобы назначить функцию в существующем приложении выберите узел приложения в **Applications tree**, нажмите кнопку зелёного плюса и выполните команду **POU...**. Как альтернатива, нажмите правой кнопкой узел **Application** и выполните команду **Add Object** → **POU** из контекстного меню. Для добавления независимой от приложения POU, выберите узел **Global** в **Applications tree** и выполните те же команды.

В диалоговом окне **Add POU** выберите опцию **Function**. Введите **Name** (<function name>) и **Return Data Type** (<data type>) для новой функции и выберите язык желаемой реализации. Для выбора типа возвращаемых данных нажмите кнопку **...** для открытия диалогового окна **Input Assistant**. Нажмите **Open** для подтверждения. Открывается вид редактора для новой функции, и Вы можете начать редактирование.

Декларирование функции

Синтаксис:
FUNCTION <function name> : <data type>

Далее следуют объявления переменных для входов и переменных функции.

Назначьте результат функции. Это означает, что имя функции используется как выходная переменная.

Не объявляйте в функции локальных переменных, таких как `RETAIN` или `PERSISTENT`, потому что это не будет иметь действия.

Компилятор генерирует соответствующие сообщения, если обнаруживаются локальные переменные, объявленные как `RETAIN` или `PERSISTENT`.

Пример функции на ST: эта функция принимает 3 входные переменные и возвращает произведение двух последних, добавленное к первой.

Вызов функции

Вызов функции на языке ST может происходить как операнд в выражениях.

На IL Вы можете располагать вызов функции только в пределах действий шага или в пределах перехода.

Функции (в противоположность функциональному блоку) не содержат информации о внутреннем состоянии, поэтому вызов функции с одними и теми же аргументами (входными параметрами) всегда производит одно и то же значение (выход). По этой причине функции не могут содержать глобальных переменных и адресов.

Пример вызова функции на IL

Вызов функции на IL;

```
LD          5
Fct         3          ,
            22
ST          result
```

Пример вызова функции на ST

```
result := fct1(5,3,22);
```

Пример вызова функции на FBD

Вызов функции на FBD:

Пример:

```
fun(formal1 := actual1, actual2); // -> error message
fun(formal2 := actual2, formal1 := actual1); // same semantics as the following:
fun(formal1 := actual1, formal2 := actual2);
```

Согласно стандарту IEC 61131-3 функции могут иметь дополнительные опции. Они должны быть назначены в вызове функции. В ST, например, согласно следующему синтаксису:

out1 => <output variable 1> | out2 => <output variable 2> | ...further output variables

Пример

Функция `fun` определена с 2 входными переменными `in1` и `in2`. Возвращаемое значение `fun` записано в локально объявленные выходные переменные *(см. стр. 515)* (`VAR_OUTPUT`) `loc1` и `loc2`.

```
fun(in1 := 1, in2 := 2, out1 => loc1, out2 => loc2);
```

Метод

Обзор

Функциональность **Method** доступна, только если выбрана в текущем используемом наборе свойств (**Options** → **Features** → **Predefined feature sets**).

Вы можете использовать методы для описания последовательности инструкций, потому что они поддерживают объектно-ориентированное программирование. В отличие от функции, метод не является независимой POU, но должен быть назначен на функциональный блок *(см. стр. 174)*. Он может рассматриваться как функция, которая содержит экземпляр соответствующего функционального блока. Как и функция, он имеет возвращаемое значение, собственную часть объявлений временных переменных и параметров.

Так же как средства объектно-ориентированного программирования, Вы можете использовать интерфейсы *(см. стр. 163)* для организации методов, доступных в проекте. Интерфейс – это коллекция методов-прототипов. Это означает, что метод, назначенный на интерфейс, содержит только часть объявления, но не реализации. Кроме объявления не разрешены локальные и статические переменные, только переменные входов, выходов и входов/выходов. Реализация метода производится в функциональном блоке, который реализует интерфейс *(см. стр. 183)* и использует метод.

ПРИМЕЧАНИЕ: При копировании и перемещении метода или свойства из POU в интерфейс, содержащиеся реализации автоматически удаляются. При копировании и перемещении из интерфейса в POU Вас попросят специфицировать желаемый язык реализации.

Вставка метода

Для назначения метода на функциональный блок или интерфейс, выберите подходящий функциональный блок или узел интерфейса в **Applications tree**, нажмите кнопку зелёного плюса и выполните команду **Method**. Альтернативно Вы можете нажать правую кнопку мыши на функциональном блоке или узле интерфейса и выполнить команду **Add Object** → **Method** из контекстного меню.

В диалоговом окне **Add Method** введите **Name,** желаемый **Return Type**, **Implementation Language** и **Access Specifier** (см. ниже). Для выбора возвращаемого типа данных нажмите кнопку **…**, чтобы открыть диалоговое окно **Input Assistant...** .

Access specifier: По причинам совместимости спецификаторы доступа являются опциями. Спецификатор **PUBLIC** доступен как эквивалент для набора без спецификатора.

Альтернативно выберите одну опцию из списка выбора:
- **PRIVATE**: Доступ к методу ограничен функциональным блоком.
- **PROTECTED**: Доступ к методу ограничен функциональным блоком и его источником.
- **INTERNAL**: Доступ к методу ограничен текущим namespace (библиотека).
- **FINAL**: Невозможен доступ к методу с перезаписью. Разрешает генерацию оптимизированного кода.

ПРИМЕЧАНИЕ: спецификаторы доступа верны для компилятора версии 3.4.4.0 и, следовательно, могут использоваться как идентификаторы в более поздних версиях. За дальнейшей информацией обращайтесь к таблице разметки версий SoMachine/CoDeSys компилятора в Руководстве пользователя по совместимости и миграции SoMachine *(см. SoMachine Compatibility and Migration, User Guide)*.

Нажмите **Open** для подтверждения. Открывается вид редактора метода.

Декларирование метода

Синтаксис:
METHOD <access specifier> <method name> : <return data type>VAR_INPUT ... END_VAR

Для описания того, как объявлять методы, поддерживаемые интерфейсами, обращайтесь к главе Интерфейс *(см. стр. 163)*.

Вызов Method

Вызовы Method также называются вызовами виртуальных функций. За дальнейшей информацией обращайтесь к разделу *Вызов метода (см. Стр. 185)*.

Помните следующее для вызова метода:

- Данные метода временные и верны только во время выполнения метода (стековые переменные). Это означает, что переменные и функциональные блоки, объявленные в методе, переинициализируются при каждом вызове метода.
- В теле метода разрешён доступ к переменным экземпляров функциональных блоков.
- Если необходимо, используйте указатель THIS *(см. стр. 189)*, который позволяет указать на текущий экземпляр.
- Переменные `VAR_IN_OUT` или `VAR_TEMP` функционального блока не доступны в методе.
- Методам, определённым в интерфейсе *(см. стр. 163)*, разрешено иметь только переменные входов, выходов и входов/выходов, но не в теле (реализующей части).
- Методы, такие как функции, могут иметь дополнительные выходы. Они должны быть назначены во время *вызова метода (см. стр. 185)*.

Специальные методы для функциональных блоков

Метод	Описание
`Init`	Метод, именуемый `FB_init`, по умолчанию объявляется неявно, но также может быть объявлен явно. Он содержит код инициализации для функционального блока, как объявлено декларационной часть функционального блока. Обращайтесь к методу `FB_init` *(см. стр. 525)*.
`Reinit`	Если метод, именуемый `FB_reinit`, объявлен для экземпляра функционального блока, он вызывается после копирования экземпляра (как во время **Online Change**) и будет реинициализировать новых экземпляр модуля. Обращайтесь к методам `FB_init`, `FB_reinit` *(см. стр. 525)*.
`Exit`	Если метод выхода, именуемый `FB_exit` желаем, он должен быть объявлен явно. Нет неявной декларации. Метод `Exit` вызывается для каждого экземпляра функционального блока перед новым вызовом, при перезапуске или онлайн изменений для всех перемещённых или удалённых экземпляров. Обращайтесь к методу `FB_exit` *(см. стр. 528)*.

Свойства *(см. стр. 161)* и свойства интерфейса *(см. стр. 167)* каждые состоят из `Set` и/или `Get` методов доступа.

Также вызов метода, когда приложение остановлено

В файле описания устройства может быть определено, что определённый метод должен всегда вызываться задачей - циклически определённым экземпляром функционального блока (или модуля библиотеки). Если этот метод имеет следующие входные параметры, он выполняется также, когда активное приложение не исполняется.

Пример

```
VAR_INPUT
pTaskInfo : POINTER TO DWORD;
pApplicationInfo: POINTER TO _IMPLICIT_APPLICATION_INFO; END_VAR
```

Программист может проверять статус приложения через `pApplicationInfo`, и может определить то, что должно произойти.

```
IF pApplicationInfo^.state = RUNNING THEN <instructions> END_IF
```

Свойство

Обзор

Функциональность **Property** (свойство) доступна, только если выбрана в текущем используемом наборе свойств (**Options** → **Features** → **Predefined feature sets**).

Свойство в расширении IEC 61131-3 доступно как средства объектно-ориентированного программирования. Оно состоит из пары методов (`Get`, `Set`). Они вызываются автоматически при доступе по чтению или записи к функциональному блоку, который имел свойство (property).

Чтобы вставить свойство (property) как объект ниже программы *(см. стр. 154)* или узла функционального блока *(см. стр. 174)*, выберите узел в **Applications tree**, нажмите кнопку зелёного плюса и выполните команду **Property**. Как альтернатива, нажмите правой кнопкой узел и выполните команду **Add Object** → **Property** из контекстного меню.

В диалоговом окне **Add Property** специфицируйте **Name**, **Return Type**, желаемый **Implementation Language**, и опционно **Access Specifier**.

Доступны те же спецификаторы, что и для методов *(см. стр. 158)*:

- **PUBLIC**
- **PRIVATE**
- **PROTECTED**
- **INTERNAL**
- **FINAL**

ПРИМЕЧАНИЕ: Свойства также могут использоваться с интерфейсами.

`Get` и `Set` Accessors для Property

2 специальных метода *(см. стр. 158)*, именуемые как accessor (аксесор - средство доступа), вставляются автоматически **Applications tree** ниже объекта свойства (property). Вы можете удалить один из них, если свойство будет использоваться только для записи или только для чтения. Аксесор, как и свойство (см. предыдущий параграф), может получить назначенный модификатор доступа в декларационной части, или через диалоговое окно **Add POU**, когда явно добавляете.

- Аксесор `Set` вызывается, когда свойство записывается, так что имя свойства используется как вход.
- Аксесор `Get` вызывается, когда свойство читается, так что имя свойства используется как вход.

Пример:

Функциональный блок `FB1` использует локальную переменную `milli`. Эта переменная определяется свойствами `Get` и `Set`:

`Get` пример:

```
seconds := milli / 1000;
```

`Set` пример

```
milli := seconds * 1000;
```

Вы можете записывать свойство функционального блока (`Set` метод), например, так `fbinst.seconds := 22;`.(`fbinst` - это экземпляр `FB1`).

Вы можете читать свойство функционального блока (`Get` метод), например, так `testvar := fbinst.seconds;`.

В следующем примере свойство `Prop1` назначено на функциональный блок `fb`:

Свойство может иметь дополнительные локальные переменные, но не дополнительные входы и по контрасту с функцией *(см. стр. 156)* или методом *(см. стр. 158)* – не дополнительные выходы.

ПРИМЕЧАНИЕ: Когда копируется или перемещается метод или свойство из POU в интерфейс, содержащиеся реализации автоматически удаляются. При копировании или перемещении из интерфейса в POU, Вас запросят специфицировать желаемый язык реализации.

Мониторинг свойства

Свойство может наблюдаться в режиме онлайн или с помощью встроенного мониторинга *(см. стр. 350)* или с помощью списка просмотра *(см. стр. 416)*. Предварительным условием для мониторинга свойства является добавление прагмы `{attribute 'monitoring:=variable'}` (обращайтесь к главе Мониторинг атрибутов *(см. стр. 556)*) вверху его определения.

Интерфейс

Обзор

Функциональность **Interface** доступна, только если выбрана в используемом в данный момент наборе свойств (**Options** → **Features** → **Predefined feature sets**).

Использование интерфейсов – средства объектно-ориентированного программирования. POU интерфейса описывает набор прототипов методов *(см. стр. 158)* и свойств *(см. стр. 161)*. Прототип означает, что содержатся только декларации, но не реализация. Интерфейс может быть описан как пустая оболочка функционального блока *(см. стр. 174)*. Он должен быть представлен *(см. стр. 183)* в объявлении функционального блока с целью получить реализацию в экземплярах функционального блока. Не являясь частью определения функционального блока, он может быть заполнен специфическим для функционального блока программным кодом. Функциональный блок может реализовывать один или несколько интерфейсов.

Один и тот же метод может быть реализован с идентичными параметрами, но различным кодом реализации с помощью различных функциональных блоков. Следовательно, интерфейс может быть использован / вызван в любой POU без необходимости того, чтобы POU идентифицировала конкретный функциональный блок, к которому имеет отношение.

Пример определения интерфейса и использования в функциональном блоке

Интерфейс `IFT` вставлен ниже приложения. Он содержит два метода `Method1` и `Method2`.

Ни интерфейс, ни методы не содержат какого-либо кода реализации. Только декларационная часть методов заполнена желаемыми определениями переменных:

Интерфейс с двумя методами:

Один или множество функциональных блоков могут теперь быть вставлены, реализовывая определённый выше интерфейс `ITF`.

Создание функционального блока, реализующего интерфейс

Когда функциональный блок `POU` добавляется в **Applications tree**, методы `Method1` и `Method2` автоматически вставляются ниже, как определено `ITF`. Здесь они могут быть заполнены специфическим для функционального блока кодом реализации.

Использование интерфейса в определении функционального блока

Интерфейс может расширять другие интерфейсы, используя `EXTENDS` (см. следующий пример «*Пример расширения интерфейса*» (*см. стр. 166*)) в определении интерфейса. Это также возможно для функциональных блоков.

Свойства интерфейса

Интерфейс также может определять свойство интерфейса, состоящее из методов аксессора `Get` и/или `Set`. За информацией о свойствах обращайтесь к разделам *Свойство (см. стр. 161)* и *Свойства интерфейса (см. стр. 167)*. Свойство в интерфейсе, также как и возможно включенные методы является только прототипом, что означает, что оно не содержит кода реализации. Как и в методах, оно автоматически добавляется в функциональный блок, который представляет интерфейс. Затем оно может быть заполнено специфическим программным кодом.

Рассмотрения

Рассмотрим следующее:

- Не разрешено объявлять переменные внутри интерфейса. Интерфейс не имеет тела (реализующей части) и действий. Только набор методов определён внутри интерфейса и этим методам разрешено иметь переменные входов, переменные выходов и переменные входов/выходов.
- Переменные, объявленные с типом интерфейса, обрабатываются как ссылки.
- Функциональный блок, реализующий интерфейс, должен иметь назначенные методы и свойства, которые поименованы точно так, как в интерфейсе. Они должны содержать идентично поименованные входы, выходы и входы/выходы.

ПРИМЕЧАНИЕ: Когда копируется или перемещается метод или свойство из POU в интерфейс, содержащиеся реализации автоматически удаляются. При копировании или перемещении из интерфейса в POU, Вас запросят специфицировать желаемый язык реализации.

Вставка интерфейса

Для добавления интерфейса в приложение, выберите узел **Application** в дереве **Applications tree или в Software Catalog** → **Assets**, нажмите кнопку зелёного плюса и выполните команду **Add Other Objects...** → **Interface**. Альтернативно Вы можете выполнить команду **Add Object** → **Interface**. Если Вы выберите узел **Global** перед выполнением команды, новый интерфейс будет доступен для всех приложений.

В диалоговом окне **Add Interface** введите имя нового интерфейса (<interface name>). Дополнительно Вы можете активировать опцию **Extends:** если Вы хотите, чтобы текущий интерфейс был расширением *(см. стр. 180)* другого интерфейса.

Пример расширения интерфейса

Если `ITF1` расширяет `ITF_base`, все методы, описанные `ITF_base`, будут автоматически доступны в `ITF1`.

Расширение интерфейса

Нажмите **Add** для подтверждения установок. Открывается вид редактора для нового интерфейса.

Декларирование интерфейса

Синтаксис
INTERFACE <interface name>
Для интерфейса, расширяющего другой интерфейс:
INTERFACE <interface name> EXTENDS <base interface name>

Пример

```
INTERFACE interface1 EXTENDS interface_base
```

Добавление желаемого набора методов

Для завершения определения интерфейса добавьте желаемый набор методов. Для этой цели выберите узел интерфейса в **Applications tree** или в **Software Catalog** → **Assets** и выполните команду **Interface method...**. Открывается диалоговое окно **Add Interface Method** для определения метода, являющегося частью интерфейса. Альтернативно выберите узел интерфейса в **Applications tree**, нажмите кнопку зелёного плюса и выберите **Interface Method**. Добавляйте сколько хотите методов и помните, что эти методы могут иметь только переменные входов, переменные выходов и переменные входов/выходов, но не тело (реализующую часть).

Свойство интерфейса

Обзор

Свойство, доступное как средства объектно-ориентированного программирования, может – среди методов и программ –использоваться также внутри определения интерфейса *(см. стр. 163)*. В этом случае оно именуется свойством интерфейса. Чтобы добавить его к интерфейсу, выбранному в **Applications tree**, нажмите кнопку зелёного плюса и выполните команду **Interface Property...**. Как альтернатива нажмите правой кнопкой узел интерфейса и выполните команду **Add Object → Interface property** из контекстного меню.

За дальнейшей информацией о свойстве и его методах обращайтесь к *Свойству (см. стр. 161)*.

Свойство интерфейса расширяет описание интерфейса. Как и интерфейс, оно только определяет, что методы `Get` и/или `Set` аксессора (Вы можете использовать оба или только один из них) принадлежат интерфейсу, однако не предоставляет кода реализации для них. Когда функциональный блок расширен интерфейсом, содержащим свойства, эти свойства и связанные с ними методы `Get` и/или `Set` аксессора автоматически вставляются в **Devices tree** ниже объекта функционального блока. Они могут редактироваться с целью добавления желаемого кода реализации.

Пример

На следующем рисунке интерфейс `NamedItem` получил свойство `Name` с методом `Get` и `Set` аксессора. Аксесор `Get` в этом примере предназначен использоваться для чтения имени какой-либо записи от функционального блока, реализующего интерфейс. Аксесор `Set` может использоваться для записи имени в этот функциональный блок. Оба метода не могут редактироваться внутри определения интерфейса, но это будет произведено позже в функциональном блоке.

Функциональный блок `FB_Name_xy1` был добавлен к **Devices tree**, реализующему интерфейс (`FUNCTION_BLOCK FB_Name_xy1 IMPLEMENTS NamedItem`). Следовательно, свойство `Name` с `Get` и `Set` методами были вставлены автоматически ниже `FB_Name_xy1`. Здесь Вы можете редактировать методы аксессора, например, путём того, как переменная `name_of_xy1` читается и, таким образом, имя элемента есть `got`. В другом функциональном блоке, также реализующем тот же интерфейс, `Get` метод может быть заполнен другим кодом. Этот код может обеспечивать имя любого другого элемента. Метод `Set` в примере используется для записи имени – определяемого программой `PLC_PRG ('abc')` – в функциональный блок `FB_Name_xy2`.

Интерфейс `NamedItem`, реализованный в 2 функциональных блоках

2 функциональных блока, реализующих интерфейс `NamedItem`

Функциональный блок `FB_Name_xy1`

```
FUNCTION_BLOCK FB_Name_xy1 IMPLEMENTS NamedItem VAR_INPUT
END_VAR VAR_OUTPUT
END_VAR VAR
    name_of_xy1: STRING:='xy1'; END_VAR
```

Функциональный блок `FB_Name_xy2`

```
FUNCTION_BLOCK FB_Name_xy2 IMPLEMENTS NamedItem VAR_INPUT
END_VAR VAR_OUTPUT
END_VAR VAR
    name_of_xy2: STRING:='xy2';
 name_got_from_PLC_PRG: STRING;
END_VAR
```

Реализация кода методов `Get` и `Set` аксесора ниже 2 функциональных блоков

FB_Name_xy1.Get

```
VAR END_VAR
name := name_of_xy1;
```

FB_Name_xy2.Get

```
VAR END_VAR
name := name_of_xy2;
```

FB_Name_xy2.Set

```
VAR END_VAR
name_got_from_PLC_PRG:=name;
```

Доступ к функциональным блокам из программы `PLC_PRG`

```
PROGRAM PLC_PRG VAR
    FBxy1_inst: FB_Name_xy1; FBxy2_inst:
    FB_Name_xy2; namexy1: STRING;
    namexy2: STRING;
END_VAR
//get name out of fb namexy1:=FBxy1_inst.Name;
namexy2:=FBxy2_inst.Name;
//write name to fb FBxy2_inst.Name:='abc';
```

Действие

Обзор

Вы можете определять действия (action) и назначать их функциональным блокам *(см. стр. 174)* и программам *(см. стр. 154)*. Действие является дополнительной реализацией. Оно может быть создано на языке, отличном от основной реализации. Каждому действию даётся имя.

Действие работает с данными функционального блока или программы, в которую оно входит. Оно использует уже определённые переменные входов/выходов и локальные переменные и не содержит своих собственных деклараций.

Пример Action (действия) в функциональном блоке

Следующая иллюстрация показывает действие в FB

В этом примере каждый вызов функционального блока `FB1` увеличивает или уменьшает выходную переменную `out` в зависимости от значения входной переменной `in`. Вызов акции `Reset` функционального блока устанавливает выходную переменную `out` в 0. Одна и та же переменная `out` записана в обоих случаях.

Вставка Action

Чтобы добавить действие, выберите соответствующую программу или функциональный блок в **Applications Tree** или в узле **Global** дерева **Applications Tree**, нажмите кнопку зелёного плюса и выполните команду **Action...**. Альтернативно нажмите правую кнопку мыши на программе или функциональном блоке узла и выполните команду **Add Object** → **Action**. В диалоговом окне **Add Action** определите для action **Name** и желаемый **Implementation Language**.

Вызов Action

Синтаксис

```
<Program_name>.<Action_name>
```

или

```
<Instance_name>.<Action_name>
```

Рассмотрим нотацию в FBD (см. следующий пример).

Если требуется вызвать действие action внутри её собственного блока, то есть программу или функциональный блок, в который оно входит, достаточно использовать имя действия.

Примеры

Этот раздел приводит примеры вызова описанного выше действия action из различных POU.

Объявление для всех примеров:

```
PROGRAM PLC_PRG VAR
    Inst : Counter; END_VAR
```

Вызов действия `Reset` в другой POU, запрограммированной на IL:

```
CAL Inst.Reset(In := FALSE) LD Inst.out
ST ERG
```

Вызов действия `Reset` в другой POU, запрограммированной на ST:

```
Inst.Reset(In := FALSE); Erg :=
Inst.out;
```

Вызов действия `Reset` в другой POU, запрограммированной на FBD:

Действие в FBD

ПРИМЕЧАНИЕ: IEC стандарт не распознаёт других действий, чем действия в языке последовательных диаграмм (SFC). Эти действия являются основной частью, содержащей инструкции, которые обрабатываются в конкретных шагах диаграммы

Внешние функция, функциональный блок, метод

Обзор

Для внешних функции, функционального блока или метода системой программирования не будет генерироваться код.

Выполните следующие шаги для создания внешнего POU:

Шаг	Действие
1.	Добавьте желаемый объект POU к узлу **Global** дерева **Applications tree** Вашего проекта, как любой внутренний объект и определите соответствующие переменные входов и выходов. **ПРИМЕЧАНИЕ:** Определите локальные переменные во внешних функциональных блоках. Не определяйте их во внешних функциях или методах. Переменные `VAR_STAT` не могут быть использованы в системе выполнения.
2.	Определите POU как внешнюю: Для этой цели нажмите правой кнопкой объект POU в узле **Global** дерева **Applications tree** и выполните команду **Properties** из контекстного меню. Откройте вкладку **Build** и активируйте опцию **External Implementation (Late link in the runtime system)**.

В системе исполнения эквивалентная функция, функциональный блок или метод будут реализованы. При загрузке программы эквивалент для каждой внешней POU ищется в системе исполнения. Если эквивалент находится, он связывается.

POUs для неявных проверок

Обзор

Ниже приложения (application) Вы можете добавить специальные POU. Они будут доступны, если неявно производимая функциональность проверок для массивов и границ диапазонов, деления на ноль и указателей должна использоваться во время выполнения. Вы можете деактивировать эту функциональность в случае устройств, в которых эти функции проверок производятся специальной явной библиотекой.

Для этой цели диалоговое окно **Add Object** → **POUs for implicit checks** обеспечивает следующие функции:

- CheckBounds *(см. стр. 592)*
- CheckDivInt *(см. стр. 625)*
- CheckDivLInt *(см. стр. 625)*
- CheckDivReal *(см. стр. 625)*
- CheckDivLreal *(см. стр. 625)*
- CheckRange *(см. стр. 599)*
- CheckRangeUnsigned *(см. стр. 599)*
- CheckPointer *(см. стр. 582)*

После того, как Вы вставите POU проверки, они открываются в редакторе, соответствующем выбранному языку реализации. Реализация по умолчанию, которую Вы можете адаптировать к Вашим требованиям, доступна в редакторе ST.

После того, как Вы вставили определённое POU проверки, опция больше не доступна в диалоговом окне, чтобы предотвратить двойную вставку. Если все типы POU проверки уже были вставлены ниже приложения, диалоговое окно **Add Object** больше не выдаёт опцию **POUs for implicit checks**.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРАВИЛЬНАЯ ФУНКЦИОНАЛЬНОСТЬ НЕЯВНЫХ ПРОВЕРОК

Не изменяйте декларационную часть неявной проверки с целью поддержки её функциональной целостности.

Отказ следовать этим указаниям может привести к травмам или повреждению оборудования.

ПРИМЕЧАНИЕ: Что касается SoMachine V4.0, после удаления функций неявной проверки (таких как CheckBounds) из Вашего приложения, не возможны **Online Change** (онлайн изменения), только загрузка. Будет появляться соответствующее сообщение.

Раздел 7.2

Функциональный блок

Содержание раздела

Этот раздел содержит следующие темы:

Тема	Страница
Общая информация	175
Экземпляр функционального блока	177
Вызов функционального блока	178
Расширение функционального блока	180
Реализация интерфейсов	183
Вызов метода	185
Указатель `SUPER`	187
Указатель `THIS`	189

Общая информация

Обзор

Функциональный блок – это POU *(см. стр. 149)*, которая обеспечивает 1 или более значений во время выполнения программы контроллером. В противоположность функции, значения выходных переменных и необходимые внутренние переменные будут сохраняться от одного выполнения функционального блока до другого.

Следовательно, вызов функционального блока с неизменными аргументами (входными параметрами) не всегда произведёт те же самые выходные значения.

В дополнение к функциональности, описанной в стандарте IEC11631-3, поддерживается объектно-ориентированное программирование, и функциональный блок может быть определён как расширение *(см. стр. 180)* других функциональных блоков. Они могут включать определения интерфейса *(см. стр. 183)*, относящиеся к вызову Метода *(см. стр. 185)*. Поэтому наследственность может быть использована при программировании с функциональными блоками.

Функциональный блок всегда вызывается через экземпляр (instance) *(см. стр. 177)*, который репродуцирует (копирует) функциональный блок.

Добавление функционального блока

Для добавления функционального блока в существующее приложение выберите соответствующий узел в **Software Catalog** → **Assets** или **Applications tree**, нажмите кнопку зелёного плюса и выберите **POU...**. Альтернативно Вы можете нажать правой кнопкой по узлу и выполнить команду **Add Object** → **POU**. Для создания функционального блока, независимого от приложения, выберите узел **Global** в **Applications tree** or **Assets**.

В диалоговом окне **Add Object** выберите опцию **Function Block**, введите для функционального блока **Name** (<идентификатор>) и выберите желаемый **Implementation Language** (язык программирования).

Дополнительно Вы можете установить следующие опции:

Опция	Описание
Extends	Введите имя другого функционального блока, доступного в проекте, который должен быть базовым для данного блока. За деталями обращайтесь к *Расширению функционального блока (см. стр. 180)*.
Implements	Введите имена интерфейсов *(см. стр. 163)*, доступных в проекте, которые должны быть представлены в текущем функциональном блоке. Вы можете ввести несколько интерфейсов, разделённых запятыми. За деталями обращайтесь к *Реализации интерфейсов (см. стр. 183)*.

Опция	Описание
Access specifier	По причинам совместимости спецификаторы доступа являются опциями. Спецификатор **PUBLIC** является эквивалентом не установленных спецификаторов. В качестве альтернативы выберите одну из опций из списка выбора: ● **INTERNAL**: Доступ к функциональному блоку ограничен текущим namespace (библиотека). ● **FINAL**: Производный доступ невозможен, то есть функциональный блок не может быть расширен другим. Разрешена генерация оптимизированного кода. **ПРИМЕЧАНИЕ:** Спецификаторы доступа действительны как версия 3.4.4.0 компилятора и, следовательно, могут использоваться также как идентификаторы в более ранних версиях. За дальнейшей информацией обращайтесь к таблице разметки версий компилятора SoMachine/CoDeSys в Руководстве пользователя по совместимости и миграции SoMachine *(см. SoMachine Compatibility and Migration, User Guide)*.
Method implementation language	Выберите желаемый язык программирования для всех объектов метода, создаваемых при реализации интерфейса, независимо от этой установки собственно для функционального блока.

Нажмите **Add** для подтверждения установок. Открывается вид редактора для нового функционального блока, и Вы можете начать редактирование.

Декларирование функционального блока

Синтаксис
FUNCTION_BLOCK <спецификатор доступа> <имя функционального блока> | EXTENDS < имя функционального блока > | IMPLEMENTS <список разделённых запятыми имён интерфейсов>

Далее следует декларация переменных.

Пример

`FBexample`, показанный на следующем рисунке, имеет 2 входных переменных и 2 выходных переменных `out1` и `out2`.

`out1` есть сумма 2 входов, `out2` есть результат сравнения на эквивалентность.

Пример функционального блока в ST

Экземпляр функционального блока

Обзор

Функциональные блоки вызываются *(см. стр. 178)* через экземпляр, который является репродукцией (копией) функционального блока *(см. стр. 175)*.

Экземпляры, как и переменные, объявляются локально или глобально. Имя функционального блока индицируется как тип данных идентификатора.

Синтаксис для декларирования экземпляра функционального блока

<идентификатор>:<имя функционального блока>;

Пример

Декларация (например, в декларационной части программы) экземпляра `INSTANCE` функционального блока `FUB`:

```
INSTANCE: FUB;
```

Декларационные части функциональных блоков и программ могут содержать декларации экземпляров. Но декларации экземпляров не разрешены в функциях.

Вызов функционального блока

Обзор

Функциональные блоки *(см. стр. 175)* вызываются через экземпляр функционального блока. Поэтому экземпляр функционального блока должен быть объявлен локально или глобально. Обращайтесь к разделу *Экземпляр функционального блока (см. стр. 177)* за информацией о том, как выполнить декларацию.

Далее желаемый функциональный блок может быть доступен при использовании следующего синтаксиса.

Синтаксис

<имя экземпляра>.<имя переменной>

Рассмотрения

- Только входные и выходные переменные функционального блока могут быть доступны с внешней стороны экземпляра функционального блока, не его внутренние переменные.
- Доступ к экземпляру функционального блока ограничен POU *(см. стр. 149)*, в которой он был объявлен, за исключением его объявления глобально.
- При вызове экземпляра желаемые значения могут быть назначены параметрам функционального блока. Смотрите следующий параграф *Назначение параметров при вызове*.
- Переменные входа / выхода (`VAR_IN_OUT`) функционального блока передаются как указатели.
- В SFC вызовы функциональных блоков могут иметь место только в шагах.
- Имя экземпляра функционального блока может быть использовано как входной параметр функции или другого функционального блока.
- Все значения функционального блока сохраняются до следующего исполнения функционального блока. Поэтому функциональный блок не всегда возвращает те же самые выходные значения, даже если произведен с идентичными аргументами.

ПРИМЕЧАНИЕ: Если, по крайней мере, одна из переменных функционального блока является остаточной (remanent), весь экземпляр сохраняется в области сохранения данных.

Примеры доступа к переменным функционального блока

Предположение: Функциональный блок `fb` имеет входную переменную `in1` типа INT. Смотрите здесь вызов этой переменной из программы `prog`. Смотрите объявления и реализацию на ST:

```
PROGRAM prog VAR
inst1:fb;
END_VAR
inst1.in1:=22;   (* fb is called and input variable in1 gets assigned value 22 *)
inst1(); (* fb is called, this is needed for the following access on the output
variable *)
res:=inst1.out1; (* output variable of fb is read *)
```

Пример вызова функционального блока на FBD:

Назначение параметров при вызове

В текстовых языках IL и ST Вы можете установить входные и/или выходные параметры немедленно, когда вызываете функциональный блок. Значения могут быть назначены параметрам в круглых скобках после имени экземпляра функционального блока. Для входных параметров это назначение производится использованием `:=` как при инициализации переменных *(см. стр. 505)* в позиции декларации. Для выходных параметров используется `=>`.

Пример вызова с аргументами

В этом примере функциональный блок таймера (экземпляр `CMD_TMR`) вызывается с назначениями для параметров `IN` и `PT`. Затем выходная переменная `Q` назначается переменной `A`. Результирующая переменная адресуется именем экземпляра функционального блока, последующей точкой и именем переменной:

```
CMD_TMR(IN := %IX5, PT := 300); A:=CMD_TMR.Q
```

Пример вставки через Input Assistant с аргументами

Если экземпляр вставляется через **Input Assistant** с опцией **With arguments** в виде реализации POU на ST или IL, он автоматически отображается согласно синтаксису, показанному в следующем примере, со всеми своими параметрами, даже если нет необходимости требования назначения этих параметров.

Для предыдущего примера вызов был бы показан следующим образом.

```
CMD_TMR(in:=, pt:=, q=>)
-> fill in, e.g.:
CMD_TMR(in:=bvar, pt:=t#200ms, q=>bres);
```

Расширение функционального блока

Обзор

Поддерживая объектно-ориентированное программирование, функциональный блок мог быть произведён из другого функционального блока. Это означает, что функциональный блок может расширять другой, следовательно, автоматически получает свойства основного функционального блока в дополнение к своим собственным.

Расширение производится использованием ключевого слова `EXTENDS` в декларации функционального блока. Вы можете выбрать опцию `EXTENDS` во время добавления функционального блока в проект через диалоговое окно **Add Object**.

Синтаксис

FUNCTION_BLOCK <имя функционального блока> EXTENDS <имя функционального блока >

Далее следует декларация переменных.

Пример

Описание функционального блока `fbA`

```
FUNCTION_BLOCK fbA VAR_INPUT
    x:int; END_VAR
...
```

Описание функционального блока `fbB`

```
FUNCTION_BLOCK fbB EXTENDS fbA VAR_INPUT
    ivar: INT := 0; END_VAR
...
```

Расширение через EXTENDS

Расширение через `EXTENDS` означает:

- `fbB` содержит все данные и методы, которые определены для `fbA`. Экземпляр `fbB` теперь может быть использован в любом контексте, где функциональный блок типа `fbA` расширен.
- `fbB` разрешено перезаписывать методы, определённые в `fbA`. Это означает: `fbB` может декларировать метод с тем же самым именем и теми же входами и выходом, что объявлено в `A`.
- `fbB` не разрешено использовать переменные функционального блока с тем же самым именем, что используются в `fbA`. В этом случае компилятор будет генерировать сообщение об ошибке.
- `fbA` переменные и методы могут быть доступны прямо внутри контекста `fbB`, используя указатель SUPER *(см. стр. 187)* (`SUPER^.<method>`).

ПРИМЕЧАНИЕ: Многократное наследование не разрешено.

Пример

```
FUNCTION_BLOCK FB_Base VAR_INPUT
END_VAR VAR_OUTPUT
    iCnt : INT; END_VAR
VAR END_VAR
THIS^.METH_DoIt(); THIS^.METH_DoAlso();

    METHOD METH_DoIt : BOOL VAR
   END_VAR
    iCnt := -1; METH_DoIt := TRUE;

    METHOD METH_DoAlso : BOOL VAR
    END_VAR
    METH_DoAlso := TRUE;
FUNCTION_BLOCK FB_1 EXTENDS FB_Base VAR_INPUT
END_VAR VAR_OUTPUT
END_VAR VAR END_VAR
// Calls the method defined under FB_1
THIS^.METH_DoIt(); THIS^.METH_DoAlso();
// Calls the method defined under FB_Base
SUPER^.METH_DoIt(); SUPER^.METH_DoAlso();
    METHOD METH_DoIt : BOOL VAR
    END_VAR
    iCnt := 1111; METH_DoIt :=
    TRUE;
```

```
PROGRAM PLC_PRG VAR
    Myfb_1: FB_1; iFB: INT;
    iBase: INT; END_VAR
Myfb_1();
iBase := Myfb_1.iCnt_Base; iFB :=
Myfb_1.iCnt_THIS;
```

Реализация интерфейсов

Обзор

С целью поддержки объектно-ориентированного программирования функциональный блок может реализовывать несколько интерфейсов *(см. стр. 163)*, которые позволяют Вам использовать методы *(см. стр. 158)*.

Синтаксис

FUNCTION_BLOCK <имя функционального блока> IMPLEMENTS <interface_1 name>|,<interface_2 name>, ..., <interface_n name>

Функциональный блок, который реализует интерфейс, должен содержать все методы и свойства (свойство интерфейса *(см. стр. 167)*) определённые в этом интерфейсе. Это включает имя, входы и выход конкретного метода или свойства, которые должны быть точно теми же самыми.

Для этой цели – когда создаётся новый функциональный блок, реализующий интерфейс – автоматически все методы и свойства, определённые в этом интерфейсе, будут вставлены ниже нового функционального блока в **Applications Tree**.

ПРИМЕЧАНИЕ: Если впоследствии методы будут добавлены к определению интерфейса, в связанных функциональных блоках автоматически они добавлены не будут. Выполните команду Implement interfaces... (реализация интерфейсов) *(см. SoMachine, Menu Commands, Online Help)*, чтобы произвести это обновление явно.

Пример

`INTERFACE I1` содержит метод `GetName`:

```
METHOD GetName : STRING
```

Функциональные блоки `A` и `B` каждый реализовывают интерфейс `I1`:

```
FUNCTION_BLOCK A IMPLEMENTS I1
FUNCTION_BLOCK B IMPLEMENTS I1
```

Следовательно, в обоих функциональных блоках метод `GetName` доступен и будет вставлен автоматически ниже каждого места, куда функциональные блоки вставляются в **Applications Tree**.

Рассмотрим декларацию переменной типа `I1`:

```
FUNCTION DeliverName : STRING VAR_INPUT
  l_i : I1; END_VAR
```

Этот вход может получить все функциональные блоки, которые реализуют интерфейс `I1`.

Пример для вызовов функций:

DeliverName(l_i := A_instance); // call with instance of type A

DeliverName(l_i := B_instance); // call with instance of type B

ПРИМЕЧАНИЕ: Переменная типа интерфейса должна получить назначение экземпляра функционального блока, прежде чем метод может быть вызван с ней. Переменная типа интерфейса всегда является ссылкой на назначенный экземпляр функционального блока.

Следовательно, вызов метода интерфейса приводит к результату вызова реализации функционального блока. Как только назначение произведено, соответствующий адрес наблюдается в режиме онлайн. Однако если ссылка ещё не назначена, значение 0 отображается внутри мониторинга в режиме онлайн.

Для этого примера смотрите в реализующей части функции `DeliverName`:

```
DeliverName := l_i.GetName(); // in this case it depends on the "real" type of l_i
whether A.GetName or B.GetName is called
```

ПРИМЕЧАНИЕ: Рассмотрите также возможность расширения функционального блока *(см. стр. 180)* использованием в декларации ключевого слова `EXTENDS`.

Вызов метода

Обзор

Объектно-ориентированное программирование с функциональными блоками – среди возможности расширения *(см. стр. 180)* через `EXTENDS` – поддерживается возможным использованием интерфейсов *(см. стр. 183)* и наследования. Это требует динамически разрешаемых вызовов методов, также называемых вызовами виртуальных функций.

Вызовы виртуальных функций требуют несколько больше времени, чем нормальный вызов функций, и используются когда:

- вызов производится через указатель на функциональный блок (`pfub`^.method)
- метод вызывается для переменной интерфейса (interface1.method)
- метод вызывает другой метод того же функционального блока
- вызов производится через ссылку на функциональный блок
- `VAR_IN_OUT` тип базового функционального блока могут быть назначены на экземпляр типа производного функционального блока

Вызовы виртуальных функций делают возможным то, что один и тот же вызов в программном коде будет вызывать различные методы во время выполнения.

Для дополнительной информации и всестороннего представления обращайтесь к:

- *Method (см. стр. 158)* для информации о методах.
- *THIS Pointer (Указатель) (см. стр. 189)* для использования указателя THIS.
- *SUPER (Указатель) (см.стр. 187)* для использования SUPER pointer.

Вызов Methods

Согласно стандарту IEC 61131-3, методы, так же как и нормальные функции *(см. стр. 156)*, могут иметь дополнительные выходы. Они могут быть назначены на вызов метода согласно синтаксиса:

<method>(in1:=<value> |, further input assignments, out1 => <output variable 1> | out2 => <output variable 2> | ...further output variables)

Это даёт такой эффект, что выход метода записывается в локально декларированные выходные переменные как данные внутри вызова.

Пример

Предположим, что функциональные блоки `fub1` и `fub2 EXTEND` функциональный блок `fubbase` и `IMPLEMENT interface1`. Содержится метод `method1`.

Возможное использование вызовов интерфейсов и методов:

```
PROGRAM PLC_PRG VAR_INPUT
  b : BOOL; END_VAR
VAR
  pInst : POINTER TO fubbase; instBase :
  fubbase;
  inst1 : fub1; inst2 : fub2;
  instRef : REFERENCE to fubbase; END_VAR
IF b THEN
  instRef REF= inst1;            (* Reference to fub1 *)
  pInst := ADR(instBase);
ELSE
  instRef REF= inst2;            (* Reference to fub2 *)
  pInst := ADR(inst1);
END_IF
pInst^.method1();                 (* If b is true, fubbase.method1 is called, else
fub1.method1 is called *)
instRef.method1();                (* If b is true, fub1.method1 is called, else
fub2.method1 is called *)
```

Предположим, что `fubbase` в верхнем примере содержит 2 метода `method1` и `method2`. `fub1` перезаписывает `method2`, но не `method1`.

`method1` вызывается, как показано в верхнем примере.

```
pInst^.method1(); (* If b is true fubbase.method1 is called, else fub1. method1 is
called *)
```

Для вызова через указатель THIS обратитесь к *Указателю THIS* *(см. стр. 189)*.

Указатель SUPER

Обзор

Для каждого функционального блока указатель с именем SUPER автоматически доступен. Он указывает на экземпляры базового функционального блока, через которые функциональный блок создан с наследованием базового функционального блока.

Это обеспечивает эффективное решение для следующего случая:

- SUPER обеспечивает доступ к методам реализации базового класса. Ключевым словом SUPER можно вызвать метод, который допустим в основном (родительском) экземпляре класса. Таким образом, не происходит динамического закрепления имени.

SUPER может использоваться только в методах и в реализации связанных функциональных блоков.

Так как SUPER является указателем на базовый функциональный блок, Вы не должны ссылаться (unreference) на него, чтобы получить адрес функционального блока: SUPER^.METH_DoIt

Вызов SUPER Call в различных языках реализации

Язык реализации	Пример
ST	SUPER^.METH_DoIt();
FBD/CFC/LD	

ПРИМЕЧАНИЕ: Функциональность SUPER ещё не реализована для Instruction List (Списка инструкций).

Пример

Локальная переменная `iVarB` перезагружает переменную `iVarB` функционального блока.

```
FUNCTION_BLOCK FB_Base
VAR_OUTPUT
    iCnt : INT; END_VAR
    METHOD METH_DoIt : BOOL
    iCnt := -1;

    METHOD METH_DoAlso : BOOL METH_DoAlso :=
    TRUE;

FUNCTION_BLOCK FB_1 EXTENDS FB_Base VAR_OUTPUT
    iBase: INT; END_VAR
// Calls the method defined under FB_1
THIS^.METH_DoIt(); THIS^.METH_DoAlso();
// Calls the method defined under FB_Base
SUPER^.METH_DoIt(); SUPER^.METH_DoAlso();
iBase := SUPER^.iCnt;

    METHOD METH_DoIt : BOOL
    iCnt := 1111;
    METH_DoIt := TRUE;

END_VAR
PROGRAM PLC_PRG
VAR
    myBase: FB_Base; myFB_1: FB_1;
    iTHIS: INT;
    iBase: INT; END_VAR
myBase();
iBase := myBase.iCnt; myFB_1();
iTHIS := myFB_1.iCnt;
```

Указатель THIS

Обзор

Для каждого функционального блока указатель с именем THIS доступен автоматически. Он указывает на экземпляр его собственного функционального блока.

Это обеспечивает эффективное решение для следующих задач:
- Если локально объявленная переменная в методе скрывает переменную функционального блока.
- Если Вы хотите обратиться к указателю его собственного экземпляра функционального блока для использования в функции.

THIS может использоваться только в методах и в реализации связанных функциональных блоков.

THIS должен быть записан прописными буквами. Иначе проверка не будет принята.

Так как THIS является указателем на наследуемый функциональный блок, Вы должны не ссылаться (unreference) на него, чтобы получить адрес этого переопределяемого (overriding) функционального блока: THIS^.METH_DoIt.

Вызов THIS в различных языках реализации

Язык реализации	Пример
ST	THIS^.METH_DoIt();
FBD/CFC/LD	

ПРИМЕЧАНИЕ: Функциональность THIS ещё не реализована для Instruction List (Списка инструкций).

Пример 1

Локальная переменная `iVarB iVarB` скрывает переменную `iVarB` фунционального блока.

```
FUNCTION_BLOCK fbA
VAR_INPUT
    iVarA: INT;
END_VAR
iVarA := 1;

FUNCTION_BLOCK fbB EXTENDS fbA VAR_INPUT
    iVarB: INT := 0;
END_VAR iVarA := 11;
iVarB := 2;

    METHOD DoIt : BOOL VAR_INPUT
    END_VAR VAR
        iVarB: INT;
    END_VAR
    iVarB := 22; // Here the local iVarB is set.
    THIS^.iVarB := 222; // Here the function block variable iVarB is set, although
iVarB is overloaded.

PROGRAM PLC_PRG VAR
    MyfbB: fbB; END_VAR
MyfbB(iVarA:=0 , iVarB:= 0); MyfbB.DoIt();
```

Пример 2

Вызов функции, который требует ссылку на свой собственный экземпляр.

```
FUNCTION funA VAR_INPUT
    pFB: fbA; END_VAR
...;

FUNCTION_BLOCK fbA VAR_INPUT
iVarA: INT;
```

```
END_VAR
...;

FUNCTION_BLOCK fbB EXTENDS fbA VAR_INPUT
   iVarB: INT := 0;
END_VAR iVarA := 11;
iVarB := 2;

    METHOD DoIt : BOOL VAR_INPUT
    END_VAR VAR
        iVarB: INT;
    END_VAR
    iVarB := 22;    //Here the local iVarB is set.
    funA(pFB := THIS^);    //Here funA is called with THIS^.

PROGRAM PLC_PRG VAR
    MyfbB: fbB; END_VAR
MyfbB(iVarA:=0 , iVarB:= 0);
MyfbB.DoIt();
```

Раздел 7.3

Объекты приложения

Содержание раздела

Этот раздел содержит следующие темы:

Тема	Страница
Data Unit Type (DUT) – Тип единицы данных	193
Global Variable List (GVL) – Список глобальных переменных	195
Global Network Variable List (GNVL) – Список сетевых глобальных переменных	197
Persistent Variables – Перманентные переменные	205
External File – Внешний файл	207
Text List – Текстовый список	209
Image Pool – Пул изображений	216

Data Unit Type (DUT) – Тип единицы данных

Обзор

Среди стандартных типов данных вы можете определить свои собственные типы данных. Вы можете создавать структуры *(см. стр. 595)*, типы перечислений *(см. стр. 597)* и ссылки *(см. стр. 585)* как data units type (DUTs – типы единиц данных) в редакторе DUT *(см. стр. 379)*.

Для описания конкретных стандартных и определяемых типов данных обращайтесь к описанию типов данных *(см. стр. 578)*.

Добавление объекта DUT

Чтобы добавить объект DUT к существующему приложению, выберите узел приложения в **Software catalog** → **Assets** или в **Applications tree**, нажмите кнопку зелёного плюса и выберите **DUT...**. Или нажмите правой кнопкой соответствующий узел и выполните команду **Add Object** → **DUT.** Чтобы создать независимый от приложения DUT объект, выберите узел **Global** в **Assets** или **Applications tree**. В диалоговом окне **Add DUT** введите имя (**Name)** для нового типа единицы данных и выберите желаемый тип **Structure**, **Enumeration**, **Alias** или **Union**.

В случае типа **Structure** Вы можете использовать принцип наследования, следовательно, поддерживая объектно-ориентированное программирование. Опционно Вы можете указать, что DUT расширяет другой DUT, который уже определён в проекте. Это означает, что определения DUT расширения будут автоматически допустимы внутри текущего DUT. Для этой цели активируйте опцию **Extends:** и введите имя другого DUT.

Нажмите **Add** для подтверждения установок. Открывается вид редактора для нового DUT, и Вы можете начать редактирование.

Декларирование объекта DUT

Синтаксис
TYPE <идентификатор> : <декларация компонентов DUT >END_TYPE

Декларация компонентов DUT зависит от типа DUT, например, структуры *(см. стр. 595)* или перечисления *(см. стр. 597)*.

Пример

Следующий пример содержит 2 DUTS, определяющих структуры `struct1` и `struct2`; `struct2` расширяет `struct1`, что означает, что Вы можете использовать `struct2.a` в Вашей реализации для доступа к переменной `a`.

```
TYPE struct1 : STRUCT
        a:INT; b:BOOL;
    END_STRUCT END_TYPE
TYPE struct2 EXTENDS struct1 : STRUCT
  c:DWORD; d:STRING;
    END_STRUCT END_TYPE
```

Global Variable List (GVL) – Список глобальных переменных

Обзор

Список глобальных переменных (GVL) используется для декларирования глобальных переменных *(см. стр. 516)*. Если GVL помещён в узел **Global** каталога **Software catalog** → **Assets** → **POUs** или **Applications tree**, переменные будут доступны для всего проекта. Если GVL назначен на конкретное приложение, переменные будут доступны в пределах приложения. **Global Variable List...**.

Для добавления GVL в существующее приложение выберите узел приложения в **Software catalog** → **Assets** → **POUs** или **Applications tree**, нажмите кнопку зелёного плюса и выберите **Global Variable List...**. Альтернативно Вы можете нажать правой кнопкой по узлу и выполнить команду **Add Object** → **Add Global Variable List...**. Если Вы выберете узел **Global** в этих видах, новый объект GVL будет независимый от приложений.

Используйте редактор GVL *(см. стр. 381)* для редактирования списка глобальных переменных.

Переменные, содержащиеся в GVL, могут быть определены, чтобы быть доступными как сетевые переменные *(см. стр. 799)* для широковещательного обмена данными с другими устройствами в сети. Для этой цели сконфигурируйте соответствующие сетевые свойства (по умолчанию в меню **View** → **Properties** → **Network Properties**) для GVL.

ПРИМЕЧАНИЕ: Максимальный размер сетевой переменной равен 255 байт. Число сетевых переменных не ограничено.

ПРИМЕЧАНИЕ: Переменные, декларированные в GVLs, инициализируются до локальных переменных POUs.

GVL для конфигурируемых констант (список параметров) в библиотеках

Значение глобальной константы, предоставляемое через библиотеку, может быть заменено значением, определяемым приложением. Для этой цели константа должна быть объявлена в списке параметров в библиотеке. Затем, когда библиотека включена в приложение, её значение может быть отредактировано во вкладке **Parameter List** менеджера **Library Manager** приложения. Смотрите следующий пример для описания того, как это сделать в деталях.

Поддержка списка параметров

Библиотека `lib1.library` обеспечивает массив переменных `g_Array`. Размер переменной массива определяется глобальной константой `g_c_Arraysize`. Библиотека включена в различные приложения, каждая требует различный размер массива. Следовательно, Вы хотите перезаписать глобальную константу библиотеки специфическим для приложения значением.

Продолжите следующим образом: Когда создаётся `lib1.library`, определите глобальную константу `g_c_Arraysize` внутри специального типа списка глобальных переменных (GVL), так называемый список параметров. Для этой цели выполните команду **Add Object** и добавьте объект списка параметров, в текущем примере именуемой `Param`. В редакторе этого объекта, который эквивалентен стандартному GVL, вставьте декларацию переменной `g_c_Arraysize`.

Список параметров `Param` в библиотеке `Lib1.library`

Редактирование параметра `g_c_Arraysize` в **Library Manager** проекта

Выберите библиотеку в верхней части **Library Manager**, чтобы получить дерево модулей. Выберите `Param` с целью открыть вкладку **Library Parameters**, показывающую декларации. Выберите ячейку в колонке **Value (editable)** и используйте свободное пространство для открытия поля редактирования. Введите желаемое новое значение для `g_c_Arraysize`. Оно будет приложено к текущему, локальному образу библиотеки после закрытия поля редактирования.

Global Network Variable List (GNVL) – Список сетевых глобальных переменных

Обзор

Функциональность **GNVL** доступна, только если выбрана в текущем используемом наборе параметров (**Options** → **Features** → **Predefined feature sets**).

Список сетевых глобальных переменных (GNVL) используется в виде **Software catalog** → **Variables** → **Global Variables** и в **Applications tree**. Он определяет переменные, которые специфицируются как сетевые переменные в другом устройстве внутри сети.

ПРИМЕЧАНИЕ: : Максимальный размер сетевой переменной равен 255 байт. Число сетевых переменных не ограничено.

Таким образом, Вы можете добавить GNVL object к приложению, если GVL *(см. стр. 195)* со специальными сетевыми свойствами (список переменных сети) доступен в 1 из других сетевых устройств. Это не зависимо от того, определено ли в том же проекте или в различных проектах. Если несколько соответствующих GVL найдены в текущем проекте для текущей сети, выберите желаемый GVL из списка выбора **Sender**, когда добавляете GNVL через диалоговое окно **Add Object** → **Add Global Network Variable List**. GVL из других проектов должны быть импортированы, как описано в этой главе.

Это означает, что GNVL в текущем устройстве (receiver - получатель) соответствует точно 1 GVL в другом устройстве (sender - отправитель).

Диалоговое окно **Add Global Network Variable List**

Описание элементов

Когда добавляется GNVL, среди **Name**, также определяется **Task**, относящаяся к поддержке сетевых переменных.

Альтернативно для прямого выбора GVL **Sender** из другого устройства Вы можете специфицировать экспортный файл *.GVL с опцией **Import from file**. Этот GVL файл был сгенерирован ранее из **Sender** GVL через диалоговое окно **View** → **Properties** → **Link To File** *(см. SoMachine, Menu Commands, Online Help)*. В любом случае необходимо, чтобы желаемый GVL был определён в другом проекте. Для этой цели выберите опцию **Import from file** в списке выбора **Sender** и введите путь к файлу в текстовом поле **Import from file** (или нажмите кнопку **...** для использования стандартного диалога для просмотра файловой системы).

Вы можете модифицировать уставки позже через диалоговое окно **View** → **Properties** → **Network Settings** *(см. SoMachine, Menu Commands, Online Help)*.

GNVL отображается редактором NVL *(см. стр. 384)*, но он не может изменяться. Он показывает текущее содержимое соответствующего GVL. Если Вы изменили основной GVL, GNVL соответственно обновляется.

Комментарий добавляется автоматически вверху декларационной части GNVL, производя информацию об отправителе (путь к устройству), имя GVL и тип протокола.

Пример списка глобальных сетевых переменных

Список глобальных сетевых переменных

ПРИМЕЧАНИЕ: Только массивы, чьи границы определены литералом или константой передаются в удалённое приложение. Выражения констант в этом случае не разрешены для определения границ.

Пример:

Example: `arrVar : ARRAY[0..g_iArraySize-1] OF INT ;` не передаётся

`arrVar : ARRAY[0..10] OF INT ;` передаётся

За дальнейшей информацией обращайтесь к главе *Сетевая коммуникация* *(см. стр. 799)*.

Пример простого обмена сетевой переменной

В следующем примере устанавливается простой обмен сетевой переменной. В контроллере отправителя создаётся список глобальных переменных (GVL). В контроллере получателя создаётся список сетевых глобальных переменных (GNVL).

Произведите следующие приготовления в стандартном проекте, где контроллер отправителя **Dev_Sender** и контроллер получателя **Dev_Receiver** доступны в **Devices tree**:

- Создайте POU (программу) **prog_sender** ниже узла **Application** устройства **Dev_Sender**.
- Под узлом **Task Configuration** этого приложения добавьте задачу **Task_S,** которая вызывает **prog_sender**.
- Создайте POU (program) **prog_rec** ниже узла **Application** устройства **Dev_Receiver**.
- Под узлом **Task Configuration** этого приложения добавьте задачу **Task_R**, которая вызывает **prog_rec**.
  ПРИМЕЧАНИЕ: 2 контроллера должны быть сконфигурированы в одной и той же подсети Ethernet сети.

Определение GVL для Sender

Шаг 1: Определите список глобальных переменных в контроллере отправителя:

Шаг	Действие	Комментарий
1	В **Software catalog** → **Assets View** → **POUs**, выберите узел **Application** контроллера **Dev_Sender** и нажмите кнопку зелёного плюса. Выберите команду **Global Variable List...**.	Отображается диалоговое окно **Add Global Variable List**.
2	Введите **Name** `GVL_Sender` и нажмите **Add** для создания нового списка глобальных переменных.	Узел **GVL_Sender** появляется ниже узла **Application** в дереве **Applications tree** и открывается редактор в средней части экрана SoMachine.

Шаг	Действие	Комментарий

3	В редакторе введите следующие определения переменных:
	`VAR_GLOBAL`
	`iglobvar:INT;`
	`bglobvar:BOOL;`
	`strglobvar:STRING; END_VAR`
	
		–

Шаг 2: Определите сетевые свойства GVL отправителя:

Шаг	Действие	Комментарий
1	В **Applications tree**, выберите узел **GVL_Sender**, нажмите кнопку зелёного плюса и выполните команду **Properties...**.	Отображается диалоговое окно **Properties - GVL_Sender**.
2	Откройте вкладку **Network properties** и сконфигурируйте параметры, как показано на изображении:	–
3	Нажмите **OK**	Диалоговое окно закрывается, и сетевые свойства GVL установлены.

Определение GNVL для получателя

Шаг 1: Определите список глобальных переменных в контроллере получателя:

Шаг	Действие	Комментарий
1	В **Applications tree**, выберите узел **Application** контроллера **Dev_Receiver**, нажмите кнопку зелёного плюса и выполните команду **Global Network Variable List...**.	Отображается диалоговое окно **Add Global Network Variable List**.
2	Сконфигурируйте параметры, как показано на изображении. 	Этот список глобальных переменных является дубликатом GVL, определённого для контроллера отправителя.

Шаг	Действие	Комментарий
3	Нажмите **Open**.	Диалоговое окно закрывается и **GNVL_Receiver** появляется ниже узла **Application** контроллера **Dev_Receiver**: Этот GNVL автоматически содержит те же самые декларации переменных, что и **GVL_Sender**.

Шаг 2: Просмотр и /или модификация сетевых установок GNVL:

Шаг	Действие	Комментарий
1	В **Devices tree**, нажмите правой кнопкой узел **GNVL_Receiver** и выберите команду **Properties...**.	Отображается диалоговое окно **Properties - GNVL_Receiver**.
2	Откройте вкладку **Network settings**.	–

Шаг 3: Протестируйте обмен сетевой переменной в онлайн режиме:

Шаг	Действие	Комментарий
1	Под узлом **Application** контроллера **Dev_Sender**, нажмите дважды POU **prog_sender**.	С правой стороны открывается редактор для **prog_sender**.

2	Введите следующий код для переменной `iglobvar`:
	prog_sender [Dev_Sender: Plc Logic: Application]
	`PROGRAM prog_sender`
	`VAR`
	`END_VAR`
	`iglobvar:=iglobvar+1;`
		–

3	Под узлом **Application** контроллера **Dev_Receiver**, нажмите дважды POU **prog_rec**.	С правой стороны открывается редактор для **prog_rec**.

4	Введите следующий код для переменной `ivar_local`:
	prog_rec [Dev_Receiver: Plc Logic: Application]
	`PROGRAM prog_rec`
	`VAR`
	`ivar_local: INT;`
	`END_VAR`
	`ivar_local:=iglobvar;`
		–

5	Войдите в приложения отправителя и получателя внутри одной и той же сети и запустите приложения.	Переменная `ivar_local` в получателе копирует значения `iglobvar`, которые в данный момент показываются в отправителе.

Persistent Variables – Перманентные переменные

Обзор

Этот объект является списком глобальных переменных, который содержит только перманентные переменные приложения. Следовательно, он должен быть назначен на приложение. Для этой цели он должен быть вставлен в **Applications tree** через выбор соответствующего узла, нажатием кнопки зелёного плюса и выбором **Add Other Objects→Persistent Variables...**.

Редактирование списка перманентных переменных в редакторе GVL *(см. стр. 381)*.
`VAR_GLOBAL PERSISTENT RETAIN` всегда предустановлена в первой линии.

Список переменных Persistent

Перманентные (Persistent) переменные переинициализируются только при **Reset (origin) <application>**. За дальнейшей информацией обращайтесь к описанию реманентных переменных *(см. стр. 518)*.
Также обращайтесь к описанию специальных команд для поддержки перманентных переменных *(см. SoMachine, Menu Commands, Online Help).*

Добавление и объявление реманентных (Remanent) переменных

Когда Вы добавляете переменные к приложению, Вы можете декларировать некоторые переменные как реманентные (сохраняющие значение) переменные. Реманентные переменные могут сохранять свои значения даже при перерывах питания, перезагрузках, перезапусках и загрузках программы приложения. Существует несколько типов реманентных переменных, объявляемых индивидуально как retain или persistent, или в комбинации как retain-persistent.

За информацией о размере памяти, резервируемом для retain и persistent переменных в различных контроллерах, обращайтесь к Руководству по программированию используемого Вами контроллера.

Для добавления списка глобальных переменных, именуемого **Persistent Variables**, в Ваше приложение проделайте следующее:

Шаг	Действие
1	Выберите соответствующий узел в **Applications tree**, нажмите кнопку зелёного плюса и выберите **Add Other Objects → Persistent Variables...**. Альтернативно Вы можете нажать правой кнопкой узел приложения и выполнить команду **Add Object → Persistent Variables...**.

Шаг	Действие
2	В диалоговом окне **Add Persistent Variables** введите имя для этого списка в текстовом поле **Name**.
3	Нажмите **Add**. **Результат:** Узел persistent переменных создан в **Applications tree**. Например, обращайтесь к параграфу *Введения* в этой главе.

Внешний файл

Обзор

Функциональность **External File** доступна, только если выбрана в используемом в настоящее время наборе параметров (**Options → Features → Predefined feature sets).**

Чтобы добавить внешний файл в узел **Global** дерева **Applications Tree** или **Tools Tree**, выберите узел **Global**, нажмите кнопку зелёного плюса и выполните команду **Add other objects → External File...**.

Нажмите кнопку **…**, чтобы открыть диалоговое окно для просмотра файла. Путь к этому файлу вводится в текстовое поле **File path**. В текстовом поле **Name** имя выбранного файла вводится автоматически без расширения. Вы можете редактировать это поле, чтобы определить другое имя для файла, под которым он должен будет поддерживаться внутри проекта.

Диалоговое окно Add External File:

Описание того, что Вы хотите сделать с внешним файлом? Секция диалогового окна

Выберете одну из следующих опций:

Опция	Описание
Remember the link.	Файл буде доступен в проекте, только если он доступен в определённом связью месте
Remember the link and embed into project.	Копия файла будет сохранена внутри проекта, но также связь с внешним файлом будет перевызвана. До тех пор, пока внешний файл будет доступен, как определено, соответственно будут реализованы представленные опции обновления. Иначе только версия файла, сохранённая в проекте, будет доступна.
Embed into project.	Только копия файла будет сохранена в проекте. В дальнейшем не будет соединения с внешним файлом.

Описание секции диалогового окна задания действий при изменении внешнего файла

Если внешний файл связан с проектом, Вы можете дополнительно выбрать одну из опций:

Опция	Описание
reload the file automatically.	Файл обновляется внутри проекта, как только он был изменён вне.
prompt whether to reload the file.	Диалоговое окно всплывает, как только файл изменён вне. Вы можете решить обновлять ли также файл внутри проекта.
do nothing.	Файл в проекте остаётся неизменным, даже когда он изменяется внешне.

Описание кнопок

Кнопка	Описание
Display file properties...	Эта кнопка открывает стандартное диалоговое окно свойств файла. Это окно появляется также, когда Вы выбираете объект файла в **Applications Tree** или **Tools Tree** и выполняете команду **Properties**. Во вкладке **External file** этого диалогового окна Вы можете просмотреть и изменять свойства.
Open	После завершения Вами установок, нажмите кнопку **Open** для добавления файла к узлу **Global** дерева **Applications Tree** или **Tools Tree**. Он открывается в том инструменте, который определён по умолчанию для данного формата файла.

Text List – Список текстов

Обзор

Функциональность **Text List** доступна, только если выбрана в используемом в настоящее время наборе параметров (**Options → Features → Predefined feature sets).**

Список текстов является объектом, управляемым глобально в узле **Global** дерева **Applications Tree** или назначенного на приложение **Applications Tree**.

Он обслуживает следующие цели:
- многоязыковую поддержку для статических *(см. стр. 210)* и динамических *(см. стр. 211)* текстов и всплывающих пояснений в визуализациях и обработке аварийных предупреждений
- обмен динамическими текстами

Текстовые списки могут быть экспортированы и (ре-) импортированы *(см. стр. 214)*. Экспорт необходим, если языковый файл в XML формате должен быть предоставлен для целевой визуализации, но также полезен для переводов.

Возможные форматы текстовых списков:
- текст
- XML

Вы можете активировать поддержку Unicode *(см. стр. 213)*.

Каждый текстовый список уникально определяется его именным пространством (namespace). Он содержит текстовые строки, которые уникально распознаются внутри списка через идентификатор (ID, состоящий из любой последовательности символов) и идентификатора языка. Используемый текстовый список специфицируется, когда конфигурируется текст для элементов визуализации.

В зависимости от языка, который установлен для визуализации, соответствующая текстовая строка отображается в онлайн режиме. Язык, используемый в визуализации, изменяется входом **Change the language**. Это достигается действием мыши, которое Вы сконфигурировали для заданного элемента визуализации. Каждый список текстов должен содержать, по крайней мере, язык по умолчанию и опционно на других языках, которые Вы выберете для определения. Если не найдено совпадений с языком, установленным в данное время в SoMachine, для списка текстов используется ввод языка по умолчанию. Каждый текст содержит форматирующие определения *(см. стр. 214)*.

Основная структура списка текстов

Идентификатор (Index)	**Default По умолчанию**	**<Language 1> <Язык 1>**	**<Language 2> <Язык 2>**	**.... <Language n> <Язык n>**
<уникальная строка символов>	<текст abc на языке по умолчанию>	<текст abc на языке 1>	<текст abc на языке 2>	...
< уникальная строка символов >	<текст xyz на языке по умолчанию>	<текст xyz на языке 1>	<текст xyz на языке 2>	...

Типы списков текстов

Есть два типа текста, используемого в элементах визуализации и, соответственно, есть два типа списков:

- `GlobalTextList` для статических текстов
- Textlist для динамических текстов

GlobalTextList для статических текстов

GlobalTextList – это специальный список текстов, где идентификаторы для конкретных текстовых записей поддерживается неявно и не редактируется. Дополнительно, список не может быть удален. Однако, список может быть экспортирован, отредактирован внешне и затем реимпортирован.

Статические тексты в визуализации, по контрасту с динамическими текстами, не обмениваются через переменные в онлайн режиме. Единственная опция для изменения языка элементов визуализации производится через вход **Change the language**. Статический текст назначается элементам визуализации через свойство **Text** или **Tooltip** в категории **Texts**. Когда первый статический текст определяется в проекте, объект списка текстов, именуемый **GlobalTextList** добавляется узел **Global** дерева **Applications Tree**. Он содержит определённую текстовую строку, находящуюся в колонке **Default** и автоматически назначается целое число в качестве текстового идентификатора. Для каждого статического текста, который создаётся позже, число в идентификаторе увеличивается на 1 и назначается на элемент визуализации.

Если статический текст введён в элемент визуализации (например, если в прямоугольнике с категорией **Texts** указана строка **Text Example)**, этот текст ищется в **GlobalTextList**.

- Если текст найден (например, **ID 4711**, **Text Example**), значение элемента **4711 TextId** будет назначено внутренней переменной. Это устанавливает связь между элементом и соответствующей линией в **GlobalTextList**.
- Если текст не найден, новая линия вставляется в **GlobalTextList** (например, **ID 4712, Text Example**). В элементе значение **4712** назначается внутренней переменной.

ПРИМЕЧАНИЕ: Если он ещё не существует, Вы можете создать список глобальных текстов явно командой **Create Global Text List**.

Если Вы имеете экспортированный, отредактированный и реимпортированный **GlobalTextList**, он проверяется на совпадение идентификаторов с теми, которые использованы в конфигурации элементов визуализации. Если необходимо, неявное обновление идентификаторов, использованных в конфигурации, будет реализовано.

Пример GlobalTextList

Создание Global Text List

GlobalTextList

ID	Standard	Deutsch	Englisch
5	%s	%s	%s
3	Deutsch	De Deutsch	German
4	Deutsch Tooltip	De Deutsch Tooltip	En German Tooltip
1	Englisch		
2	Englisch Tooltip		
0	Inkrement		

Textlist для динамических текстов

Динамические тексты могут изменяться динамически в онлайн режиме. Текстовый индекс(**ID**), который является строкой символов, должен быть уникальным внутри списка текстов. В отличие от **GlobalTextLists** Вы должны определить его. Также по контрасту с **GlobalTextLists** создавайте списки текстов для динамических текстов явно, выбрав узел **Global**, нажав кнопку зелёного плюса и выполненив команду **Add other objects** → **Text List...**.

Доступные в данное время списки динамических текстов назначаются, когда конфигурируется элемент визуализации через свойство **Dynamic texts** / **Text list**. Если Вы специфицируете имя списка текста, комбинированно с текстовым индексом (**ID**) – который может быть введён непосредственно или введением переменной проекта, которая определяет строку ID – текущий текст может быть изменён в онлайн режиме.

Список динамических текстов должен быть экспортирован, если он необходим как языковый файл для переключения языков в целевой визуализации. Специфицируйте путь к файлу в **Visualization Options**. Такой как **GlobalTextList** список динамических текстов может быть также экспортирован для внешнего редактирования и реимпортирован. В отличие от **GlobalTextList,** когда Вы импортируете списки динамических текстов, нет автоматической проверки и обновление идентификаторов.

ПРИМЕЧАНИЕ
НЕПРЕДНАМЕРЕННАЯ МОДИФИКАЦИЯ ИДЕНТИФИКАТОРОВ
Не модифицируйте идентификаторы, когда редактируете экспортированный список.
Отказ следовать этим инструкциям могут привести к повреждению оборудования.

Пример списка динамических текстов, названного ErrorList

Пример ErrorList

Детальный пример

Этот пример объясняет то, как конфигурируется элемент визуализации, который отображает соответствующее сообщение, когда обнаруживается ошибка в приложении, которое обрабатывает события ошибок, идентифицируемых через числовые ID, назначенные целой переменной `ivar_err`.

Произведите динамический текстовый список, именуемый **ErrorList**, где текстовые сообщения для ID ошибок от 0 до 4 определены на языках **German**, **English** и **Default**:

ErrorList

ID	Default	Deutsch	Englisch
0	This is Error 0. Do the following...	Fehler 0. Führen Sie folge...	Error 0. Do the...
1	This is Error 1. Close...	Fehler 1. Schließen Sie...	Error 1. Close the...
2	This is Error 2. Perform a...	Fehler 2. Führen Sie einen...	Error 2. Perform...
3	This is Error 3. Try to...	Fehler 3. Versuchen Sie...	
4	This is Error 4. Start...	Fehler 4. Starten Sie...	

Для использования IDs ошибки в конфигурации визуализации, определите переменную STRING, например `strvar_err`. Чтобы назначить целое значение `ivar_err` to `strvar_err`, используйте `strvar_err:=INT_TO_STRING(ivar_err);`.

`strvar_err` может быть введено как параметр **Textindex** в конфигурации свойств **Dynamic texts** визуализационного элемента. Этот элемент будет отображать соответствующее сообщение в онлайн режиме.

Следующий пример для обработки ID ошибки использует переменные проекта и конфигурацию элемента визуализации (**Properties**), который будет отображать соответствующее сообщение:

ErrorList / PLC_PRG

```
5     bvar: BOOL;
6     ivar_err:INT;
7     strvar_err:STRING;
8     b_err: BOOL;
9 END_VAR
```

```
1 IF b_err:=TRUE THEN
2     ivar_err:=2;
3 END_IF
4 strvar_err:=INT_TO_STRING(ivar_err);
5
```

Properties

Filter | Sort by | Sort order

Property	Value
Dynamic texts	
Textlist	'ErrorList'
Textindex	PLC_PRG.strvar_err
Tooltipindex	
Font variables	
Fontname	

m_pDynamicText..stTextIndex

Создание списка текстов

- Для создания списка текстов для динамических текстов *(см. стр. 211)*, добавьте объект **Text List** в проект **Applications Tree**. Для создания списка текстов, специфицированных для приложения, выберите узел приложения. Для создания глобального списка текстов, выберите узел **Global**. Затем нажмите кнопку зелёного плюса выбранного узла и выполните команду **Add other objects** → **Text List...**. Когда Вы специфицируете имя списка и подтвердите диалоговое окно **Add Textlist**, новый список вставится ниже выбранного узла и откроется редактор списка текстов.
- Для получения списка текстов для статических текстов *(см. стр. 210)* (**GlobalTextList**), или назначьте текст в свойстве **Text** в категории **Texts** объекта визуализации, чтобы получить список, создаваемый автоматически, или генерируйте его явно командой **Create Global Text List**.
- Чтобы открыть существующий список текстов для редактирования, выберите объект списка в **Applications Tree** или узле **Global** дерева **Applications Tree**. Нажмите правой кнопкой узел списка текстов и выполните команду **Edit Object**, или дважды нажмите узел списка текстов. Обращайтесь к таблице *Базовые структуры списков текстов* за тем, как структурирован список текстов.
- Для добавления нового текста по умолчанию в список текстов или используйте команду **Insert Text**, или редактируйте соответствующее поле в свободной линии списка. Для редактирования текстового поля списка нажмите поле, чтобы выбрать его, и затем нажмите поле снова или нажмите SPACE (пробел), чтобы получить доступ к фрейму редактирования. Введите желаемые символы и закройте фрейм регистрации нажатием RETURN.

Поддержка формата Unicode

Для использования формата Unicode активируйте соответствующую опцию в **Visualization Manager**. Кроме того, установите специальные директивы компиляции для приложения: выберите приложение в **Devices Tree**, откройте диалоговое окно **Properties**, вкладку **Build**. В поле **Compiler defines** введите `VISU_USEWSTRING`.

Диалоговое окно с определениями компилятора

Экспорт и импорт списков текстов

Списки статических и динамических текстов могут быть экспортированы как файлы в CSV формате. Экспортированные файлы также могут использоваться для внешнего добавления текстов, например с помощью внешнего транслятора. Однако, только файлы, доступные в текстовом формате (*.csv), могут быть реимпортированы.

Смотрите описание соответствующих команд списка текстов *(см. SoMachine, Menu Commands, Online Help)*.

Укажите папку, в которой экспортируемые файлы должны быть сохранены в диалоговом окне **File** → **Project Settings** → **Visualization.**

Форматирование текстов

Текст может содержать определения форматирования (`%s`, `%d`,...), которые позволяют включить текущие значения переменных в текст. Для возможных строк форматирования смотрите часть *Visualization* в онлайн справке SoMachine.

Когда используется текст со строками форматирования, замена производится в следующем порядке:
- Текущая используемая текстовая строка ищется по имени списка и ID.
- Если текст содержит определения форматирования, они заменяются на значения соответствующих переменных.

Последующее предоставление переведенных текстов

При вставке *GlobalTextList.csv* в директорию, которая используется для загрузки файлов, возможна последующая вставка оттранслированных текстов. Когда начинается загрузка проекта, фирменное программное обеспечение обнаруживает, что доступны дополнительные файлы. Текст сравнивается с тем, который есть в существующих файлах списков текстов. Затем новые и изменённые тексты применяются к файлам списков текстов. Обновлённые файлы списков текстов будут затем использованы при следующем запуске.

Компоненты списка для ввода текста

Через диалоговое окно **Tools** → **Options** → **Visualization** Вы можете специфицировать текстовый файл шаблона. Все текста колонки **Default** этого файла будут копироваться в список, который будет использоваться для функциональности **List Components**. Может быть использован файл шаблона, созданный ранее через команду **Export**.

Работа нескольких пользователей

При использовании управления исходной информацией возможно, что несколько пользователей работают одновременно над одним и тем же проектом. Если статический текст изменяется в элементах визуализации более чем одним пользователем, это вызовет изменение в **GlobalTextList** (обращайтесь к **GlobalTextList** *(см. стр. 210)*). В этом случае Text Ids могут быть больше не когерентными с элементами визуализации. Используйте следующие методы обнаружения и коррекции ошибок:
- Используйте команду **Check Visualization Text Ids**, которая может обнаружить такие ошибки в визуализации.

Используйте команду **Update Visualization Text Ids**, эти ошибки могут быть исправлены автоматически. Затронутая визуализация, а также **GlobalTextList** должны иметь разрешение записи.

Использование Textlists для изменения языка в визуализации

Если доступен соответствующий textlist, то есть textlist определяет несколько версий языка для текста, тогда язык, используемый в текстах в визуализации, может переключаться в онлайн режиме с помощью входа визуализационного элемента. Свойства **Dynamic Texts** элемента должны специфицировать используемый textlist и входное действие **OnMouse.. , Change the language**, должно быть сконфигурировано, указывая язык, который должен использоваться после произведения действия мышью.

ПРИМЕЧАНИЕ: Язык должен быть специфицирован точно такой же строкой, которая показана в заголовке колонки соответствующего списка текстов.

Пул изображений

Обзор

Функциональность **Image Pool** доступна, только если выбрана в используемом в настоящее время наборе параметров (**Options → Features → Predefined feature sets).**

Пулы изображений – это таблицы, определяющие путь файла, объект предварительного просмотра, и строковый ID для каждого изображения. С помощью спецификации ID и (для уникального доступа) дополнительно имени файла изображение может быть указано, например, когда вставляется в визуализацию (для конфигурации свойств элементов – изображений обращайтесь к *Использованию изображений, которые управляются в пулах изображений (см. стр. 217))*.

ПРИМЕЧАНИЕ: Рекомендуется уменьшить, как только можно, размер файла изображения прежде добавления его в пул изображений. Иначе размер проекта и усилия по загрузке и хранению приложений визуализации, включая изображения, могут стать большие.

Структура пула изображений

Пример пула изображений

Элемент	Описание
ID	Строковый ID (например, **logo**, **y_icon**, **2**); Уникальная ссылка на изображение достигается комбинацией имени в списке и ID (например, **List1.basic_logo**).
File name Имя файла	Путь к файлу изображения (например, *C:\programs\images\logo.bmp*)
Image Изображение	Image для предварительного просмотра изображения

Создание и редактирование Image Pool

Проект может содержать несколько пулов изображений.

Если пул ещё недоступен в проекте, тогда – когда Вы добавите первый элемент изображения и введёте ID (статический ID) для соответствующего изображения в свойствах визуальных элементов – пул изображений с именем по умолчанию **GlobalImagePool** создаётся автоматически. Запись для изображения вставляется. **GlobalImagePool** – глобальный пул, который просматривается первым, когда используется файл изображения. Кроме этого пула могут использоваться дополнительные, индивидуально названные пулы.

Чтобы создать пулы изображений вручную, проделайте следующее: **GlobalImagePool** создаётся командой **Image Pool Editor Commands** → **Create Global Image Pool**. Вы можете вставить другие объекты пула ниже узла приложения или ниже узла **Global** дерева **Applications tree** нажатием кнопке зелёного плюса и выполнением команд **Add other objects** → **Image Pool...**. В диалоговом окне **Add Image Pool** определите **Name** для пула.

Для добавления изображения вручную к пулу или используйте команду **Insert Image**, или заполните таблицу пула вручную. Для последнего выберите поле ID первой пустой линии в таблице пула, нажмите клавишу пробела, чтобы открыть фрейм редактирования и введите ID (строку). ID автоматически будет сделан уникальным. Затем поместите курсор в поле имени **File**, нажмите клавишу пробела и нажмите кнопку **…** для открытия диалогового окна **Select image**. Здесь Вы можете специфицировать путь к желаемому файлу изображения.

Использование изображений, которые управляются в пуле изображений

Если ID изображения используется в нескольких пулах изображений:

- порядок поиска: Если Вы выбрали изображение, управляемое в **GlobalImagePool**, Вам не понадобится специфицировать имя пула. Порядок поиска изображений соответствует такому для глобальных переменных:
  1. **GlobalImagePool**
  2. пулы изображений, назначенные на активное в данный момент приложение
  3. пулы изображений в узле **Global** дерева **Applications tree** кроме **GlobalImagePool**
  4. пулы изображений в библиотеках
- уникальный доступ: Вы можете напрямую вызвать желаемое изображение добавлением имени пула изображений перед ID, согласно синтаксису: <pool name>.<image ID> (например, см. `imagepool1.drive_icon` в предыдущем рисунке.)

Использование изображения в элементе визуализации типа Image

При вставки элемента изображения в визуализацию Вы можете определить - будет это статическое изображение или динамическое изображение. Динамическое изображение может быть изменено в онлайн режиме согласно значению переменной проекта.

Статические изображения:

В конфигурации элемента (свойство **Static ID**) введите ID изображения или имя пула изображений + ID изображения. Рассмотрите в этом контексте замечания в предыдущем параграфе к порядку поиска и уникальному доступу.

Динамические изображения:

В конфигурации элемента (свойство **Bitmap ID variable**) введите переменную, которая определяет ID, например, `PLC_PRG.imagevar`.

Использование изображения для визуализации фона

В определении фона визуализации Вы можете определить изображение, отображаемое как фон визуализации. Файл изображения может быть специфицирован, как описано ранее для элемента визуализации по имени пула изображений и имени файла изображения.

Раздел 7.4

Приложения

Приложения

Обзор

Приложение – это набор объектов, которые необходимы для работы конкретного экземпляра программы контроллера на определённом аппаратном устройстве (контроллере). Для этой цели независимые объекты, управляемые в узле **Global** дерева **Applications tree,** инстанцируются и назначаются на устройство. Это означает концепцию объектно-ориентированного программирования. Однако Вы можете также использовать чисто специфические для приложения POU.

Приложение представляется объектом приложения в **Applications tree**. Ниже входа в приложение вставьте объекты, определяющие набор ресурсов приложения.

По одному приложению доступно для каждого контроллера. Невозможно добавлять дальнейшие приложения.

Часть каждого приложения – это **Task Configuration**, контролирующая выполнение программы (экземпляры POU или специфические для приложения POU). Дополнительно она может иметь назначенные объекты ресурсов, как списки глобальных переменных, библиотеки и т.п. Они - в отличие от тех, которые организованы в узле **Global** дерева **Applications tree** – могут быть использованы только конкретным приложением и его дочерними. За правилами обращайтесь к описанию упорядочивания и конфигурирования объектов в **Devices tree** *(см. стр. 41)*.

Рассмотрения

При переходе к логину (входу в) приложение на целевом устройстве (контроллере или симуляторе целевого устройства) производятся две проверки: Какое приложение в данный момент находится в контроллере? Совпадают ли параметры приложения в контроллере с такими же в SoMachine? Соответствующие сообщения указывают на несовпадения и предлагают некоторые пути продолжения в этом случае. Также Вы имеете возможность удалить приложение в контроллере. Обращайтесь к описанию команды ***Login*** *(см. стр. 228)* за деталями.

Глава 8

Конфигурация задач

Содержание этой главы

Эта глава содержит следующие темы:

Тема	Страница
Конфигурация задач	220
Добавление задач	221

Конфигурация задач

Обзор

Task Configuration определяет одну или несколько задач для управления выполнением программы приложения.

Это объект ресурсов для приложения *(см. стр. 218)*. Она должна быть вставлена в **Applications tree** ниже узла приложения. Задача может вызывать POU специфической для приложения программы, которая доступна в **Applications tree** только ниже приложения. Она может также вызывать программу, которая управляется в узле **Global** дерева **Applications tree**. В последнем случае программа, которая доступна глобально, будет инстанцирована приложением.

Вы можете редактировать конфигурацию задач в **Task Editor** *(см. стр. 405)*.

В онлайн режиме **Task Editor** обеспечивает вид мониторинга, дающий информацию о циклах, временах циклов и статусе задач.

Как дополнительная функциональность конфигурации задач, если это поддерживается устройством, вид мониторинга позволяет динамический анализ POU, которые управляются задачей. Он поставляет информацию о временах циклов, количестве вызовов функциональных блоков и неиспользованных строк кода.

Добавление задач

Введение

Вы можете добавлять задачи в Ваше приложение через **Applications tree**.

Процедура

Шаг	Действие
1	В **Applications tree** выберите узел **Task Configuration**, нажмите кнопку зелёного плюса и выполните команду **Task...**. Альтернативно Вы можете нажать правой кнопкой узел **Task Configuration** и выбрать **Add Object** → **Task...** из контекстного меню. **Результат:** Открывается диалоговое окно **Add Task**.
2	В диалоговом окне **Add Task** введите имя в **Name:** текстовое поле. **Примечание:** Имя не должно содержать пробелов и не должно превышать по длине 32 символов.
3	Нажмите **Add**.

Глава 9

Управление приложениями

Содержание этой главы

Эта глава содержит следующие разделы:

Раздел	Тема	Страница
9.1	Общая информация	224
9.2	Построение и загрузка приложений	226
9.3	Выполнение приложений	242
9.4	Обслуживание приложений	243

Раздел 9.1

Общая информация

Введение

Введение

Чтобы выполнять приложение, Вы должны сначала соединить ПК с контроллером, затем загрузить приложение в контроллер.

ПРИМЕЧАНИЕ: Из-за ограничения размера памяти некоторые контроллеры не способны хранить исходную программу приложения, а только построенное приложение, которое выполняется. Это означает, что Вы не можете выгрузить исходную программу приложения из контроллера в ПК.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННОЕ ДЕЙСТВИЕ ОБОРУДОВАНИЯ

- Подтвердите, что Вы ввели верное обозначение устройства или адрес устройства в диалоге **Communication Settings**, когда загружаете приложение.
- Подтвердите, что защиты и теги машины на месте, так что любое потенциальное непреднамеренное действие машины не приведет к повреждению оборудования или травмам.
- Прочитайте и уясните всю пользовательскую документацию на программное обеспечение и связанные устройства, а также документацию, касающуюся работы оборудования или машины.

Отказ следовать этим указаниям может привести к смерти, серьёзным травмам или повреждению оборудования.

Предварительные условия

Проверьте, что Ваше приложение удовлетворяет следующим условиям перед загрузкой его в контроллер:

- активный путь установлен для верного контроллера,
- приложение, которое Вы хотите загрузить, активное,
- приложение свободно от ошибок компиляции.

Загружаемое приложение

Загружаемое приложение – приложение, которое подготовлено к запуску на контроллере. Это приложение сохранено в памяти контроллера. Для конфигурации загрузки построенного приложения нажмите правой кнопкой узел **Application** в дереве **Applications tree** и выберите команду **Properties**.

В конце успешной загрузки нового приложения отображается сообщение, спрашивающее о Вашем желании создать загружаемое приложение (boot application).

Вы можете создать загружаемое приложение вручную следующими путями:

- В офлайн режиме: Нажмите **Online** → **Create boot application** чтобы сохранить загружаемое приложение в файле.
- В онлайн режиме с контроллером в режиме STOP: Выполните команду **Online** → **Create boot application** *(см. SoMachine, Menu Commands, Online Help)*, чтобы загрузить загружаемое приложение в контроллер.

Раздел 9.2

Построение и загрузка приложений

Содержание раздела

Этот раздел содержит следующие темы:

Тема	Страница
Построение приложений	227
Логин	228
Процесс построения при изменённых приложениях	231
Загрузка приложения	232

Построение приложений

Обзор

SoMachine обеспечивает различные процедуры построения в меню Build *(см. SoMachine, Menu Commands, Online Help)*. Эти процедуры обеспечивают обработку синтаксических проверок или только изменённых объектов, или всех объектов в активном приложении.

Вы можете произвести офлайн генерацию кода с целью проверки ошибок компиляции до загрузки кода в устройство. При успешном логине генерация кода должна быть завершена без обнаружения каких-либо ошибок.

Генерация кода, компиляционная информация

Машинный код не будет генерироваться пока проект Application *(см. стр. 218)* загружен в целевое устройство (контроллер, симулятор целевого устройства). При каждой загрузке компиляционная информация, содержащая код и ссылочный ID загруженного приложения, сохраняется в директории проекта в файле *<projectname>.<devicename>.<application ID>.compileinfo*. Файл *compileinfo* удаляется, когда выполняются команды **Clean** или **Clean all**.

Генерация кода не производится, когда проект компилируется командами построения (build) (по умолчанию в меню **Build**). Процесс построения проверяет проект с целью обнаружения ошибок программирования. Любые обнаруженные ошибки программирования отображаются в виде **Messages** (категория **Build** сообщений).

Во время генерации кода дополнительные ошибки могут быть обнаружены и отражены. Эти ошибки могут быть обнаружены только генератором кода или они вызваны размещением в памяти.

Логин

Обзор

Команда **Online** → **Login** соединяет приложение с целевым устройством (контроллером или симулятором) и, следовательно, изменяет режим на онлайн.

«Быстрая» клавиша по умолчанию ALT + F8.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННОЕ ДЕЙСТВИЕ ОБОРУДОВАНИЯ

- Подтвердите, что Вы ввели верное обозначение устройства или адрес устройства в диалоге «Communication Settings», когда загружаете приложение.
- Подтвердите, что защиты и теги машины на месте, так что любое потенциальное непреднамеренное действие машины не приведет к повреждению оборудования или травмам.
- Прочитайте и уясните всю пользовательскую документацию на программное обеспечение и связанные устройства, а также документацию, касающуюся работы оборудования или машины.

Отказ следовать этим указаниям может привести к смерти, серьёзным травмам или повреждению оборудования.

Если есть онлайн управление пользователями (также обращайтесь к главе *Пользователи и группы* (*см. стр. 872*)), установленное в целевом устройстве, при логине Вам предложат ввести соответствующие имя пользователя и пароль. Для этой цели открывается диалоговое окно **Device User Logon**.

Процедуры логина

Возможны 2 различных логин процедуры, зависящие от выбранного режима **Dial-up**, в диалоговом окне **Project** → **Project Settings** → **Communication settings**. Уставка по умолчанию этого **Dial-up** режима зависит от версии SoMachine. Редактор устройства обеспечивает индивидуальные диалоги для каждого типа логина.

Тип логина	Режим Dial-up	Уставка по умолчанию для версии SoMachine	Диалог редактора устройства
1	**IP address**	V4.0 и позже	**Controller Selection** *(см. стр. 97)*
2	**active path**	V3.1 и ранее	**Communication Settings** *(см. стр. 111)*

Логин процедура с Dial-up режимом IP Address

Это логин процедура по умолчанию для версии SoMachine V4.0 и более поздних. Режим **Dial-up** в диалоговом окне **Project** → **Project Settings** → **Communication settings** установлен на **Dial up via "IP-address"**.

Для успешного логина генерация кода должна быть завершена без обнаружения ошибок (обращайтесь к главе *Процесс построения перед логином* (*см. стр. 231*)).

Шаг	Действие
1	Выполните команду **Online** → **Login** или нажмите кнопку **Login** на панели инструментов или нажмите ALT + F8. **Результат**: Так как заранее не был установлен никакой целевой адрес, открывается вид **Controller selection** редактора устройства. Окно сообщения отображается, указывая, что достоверный адрес не был определён.
2	Если только один контроллер был обнаружен сканированием SoMachine сети Ethernet, этот контроллер помечается в списке и он используется как целевое устройство. Если обнаружены несколько контроллеров, дважды нажмите контроллер, в который Вы хотите войти.
3	Выполните команду **Online** → **Login** или нажмите кнопку **Login** на панели инструментов или нажмите ALT + F8. **Результат**: Отображается окно сообщений, чтобы информировать Вас о потенциальных опасностях.
4	Нажмите **Cancel** для прекращения логин операции или нажмите ALT + F для подтверждения сообщения и логина в выбранный контроллер. **Результат**: Если Вы нажмёте ALT + F, соединение с контроллером установлено, и Вы можете загрузить приложение (*см. стр. 232*).

Логин процедура с Dial-up режимом Active Path

Это логин процедура по умолчанию для версии SoMachine V3.1 и ранних версий. Режим **Dial-up** в диалоговом окне **Project** → **Project Settings** → **Communication settings** установлен на **Dial up via "active path"**. Для успешного логина генерация кода должна быть завершена без обнаружения ошибок (обращайтесь к главе *Процесс построения перед логином* (*см. стр. 231*)). Кроме того, коммуникационные установки (*см. стр. 97*) устройства должны быть сконфигурированы корректно.

Шаг	Действие
1	Выполните команду **Online** → **Login** или нажмите кнопку **Login** на панели инструментов или нажмите ALT + F8. **Результат**: Так как заранее не был установлен никакой целевой адрес, открывается вид **Communication Settings** редактора устройства. Окно сообщения отображается, указывая, что достоверный адрес не был определён, и производится сканирование сети.
2	Если только один контроллер был обнаружен сканированием SoMachine сети Ethernet, этот контроллер помечается в списке, и он используется как целевое устройство. Если обнаружены несколько контроллеров, дважды нажмите контроллер, в который Вы хотите войти. **ПРИМЕЧАНИЕ:** Только те контроллеры перечисляются , которые имеют такой же **Target ID** как выбранный контроллер. Для отображения всех контроллеров в списке установите критерий **Filter** на **None**.

Шаг	Action
3	Выполните команду **Online** → **Login** или нажмите кнопку **Login** на панели инструментов или нажмите ALT + F8. **Результат**: Отображается окно сообщений, чтобы информировать Вас о потенциальных опасностях.
4	Нажмите **Cancel** для прекращения логин операции или нажмите ALT + F для подтверждения сообщения и логина в выбранный контроллер. **Результат**: Если Вы нажмёте ALT + F, соединение с контроллером установлено, Вы можете загрузить приложение *(см. стр. 232)*.

Процесс построения при изменённых Applications

Процесс построения перед логином

Перед **Login** и если текущий затронутый проект приложения не был скомпилирован, начиная с открытия или начиная с последнего изменения, он будет скомпилирован. Это означает, что проект будет построен соответственно выполнением **Build** в офлайн режиме и код компиляции для контроллера будет сгенерирован.

Если ошибки обнаруживаются во время компиляции, открывается окно сообщений со следующим текстом: **There are compile errors. Do you want to login without download?** (Есть ошибки компиляции. Вы хотите логин без загрузки?) Вы можете выбрать сначала исправление обнаруженных ошибок или логин, несмотря на это. В последнем случае Вы войдёте в ту версию приложения, которая, возможно, уже доступна в контроллере.

Обнаруженные ошибки перечислены в виде **Messages** (категория **Build**).

Загрузка приложения

Введение

Для выполнения приложения вначале соедините ПК с контроллером, затем загрузите приложение в контроллер.

Загрузка проекта позволяет Вам скопировать текущий проект из SoMachine в память контроллера.

ПРИМЕЧАНИЕ: Из-за ограничения размера памяти некоторые контроллеры не способны хранить исходную программу приложения, а только построенное приложение, которое выполняется. Это означает, что Вы не можете выгрузить исходную программу приложения из контроллера в ПК.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННОЕ ДЕЙСТВИЕ ОБОРУДОВАНИЯ

- Подтвердите, что Вы ввели верное обозначение устройства или адрес устройства в диалоге **Communication Settings,** когда загружаете приложение.
- Подтвердите, что защиты и теги машины на месте, так что любое потенциальное непреднамеренное действие машины не приведет к повреждению оборудования или травмам.
- Прочитайте и уясните всю пользовательскую документацию на программное обеспечение и связанные устройства, а также документацию, касающуюся работы оборудования или машины.

Отказ следовать этим указаниям может привести к смерти, серьёзным травмам или повреждению оборудования.

Предварительные условия

Проверьте, что Ваше приложение удовлетворяет следующим условиям перед загрузкой его в контроллер:

- активный путь установлен для верного контроллера,
- приложение, которое Вы хотите загрузить, активное,
- приложение свободно от ошибок компиляции.

Загружаемое приложение

Загружаемое приложение – приложение, которое подготовлено к запуску на контроллере. Это приложение сохранено в памяти контроллера. Для конфигурации загрузки построенного приложения нажмите правой кнопкой узел **Application** в дереве **Applications tree** и выберите команду **Properties**.

В конце успешной загрузки нового приложения отображается сообщение, спрашивающее о Вашем желании создать загружаемое приложение (boot application).

Вы можете создать загружаемое приложение вручную следующими путями:

- В офлайн режиме: Нажмите **Online** → **Create boot application,** чтобы сохранить загружаемое приложение в файле.

В онлайн режиме с контроллером в режиме STOP: Выполните команду **Online** → **Create boot application** *(см. SoMachine, Menu Commands, Online Help)* чтобы загрузить загружаемое приложение в контроллер.

Режимы работы

Метод загрузки отличается в зависимости от соотношений между загруженным приложением и приложением, которое Вы хотите загрузить. Четыре случая:

- Случай 1: Приложение в контроллере одинаковое с тем, которое Вы хотите загрузить. В этом случае загрузки не происходит, Вы только соединяете SoMachine с контроллером.
- Случай 2: Модификации были сделаны в приложении SoMachine по сравнению с тем, которое загружено в контроллер. В этом случае Вы можете специфицировать, что Вы хотите загрузить всё или часть изменённого приложения или сохранить приложение в контроллере неизменным.
- Случай 3: Отличающаяся или новая версия приложения уже доступна в контроллере. В этом случае Вас спросят, должно ли это приложение быть заменено.
- Случай 4: Приложение в контроллере ещё не доступно. В этом случае Вас попросят подтвердить загрузку.

Загрузка Вашего приложения в контроллер: Случай 1

Приложение в контроллере одинаковое с тем, которое Вы хотите загрузить. В этом случае загрузки не происходит, Вы только соединяете SoMachine с контроллером.

Шаг	Действие
1	Для подключения к контроллеру выберите **Online** → **Login to 'Application[YourApplicationName; Plc Logic]'**.
2	Вы подключены к контроллеру.

Загрузка Вашего приложения в контроллер: Случай 2

Модификации были сделаны в приложении SoMachine по сравнению с тем, которое загружено в контроллер.

Шаг	Действие
1	Для подключения к контроллеру выберите **Online** → **Login to 'Application[YourApplicationName; Plc Logic]'**.
2	В случае, если Вы изменили Ваше приложение и Вы хотите перезагрузить его в контроллер, появляется следующее сообщение: **Login with online change** Только изменённые части уже работающего проекта перезагружаются в контроллер. **Login with download** Всё изменённое приложение перезагружается в контроллер. **Login without any change** Изменения не загружаются. **ПРИМЕЧАНИЕ:** Если Вы выбрали опцию **Login without any change**, изменения, которые Вы сделали в приложении SoMachine, не загружаются в контроллер. В этом случае строка информации и статуса в SoMachine будут показывать **RUN** как в рабочем состоянии и будет показывать **Program modified (Online change)**. Это отличается от опций **Login with online change** или **Login with download**, когда строка информации и статуса показывает **Program unchanged**. В этом случае мониторинг переменных возможен, но логический поток может быть перепутан, потому что значения на выходах функциональных блоков могут не соответствовать значениям входов. **Примеры** В LD состояние контактов наблюдается на основе влияющих переменных. Это может иметь такой эффект, что голубой анимированный контакт, с последующей голубой связью (означающей истину) отображаются, хотя катушка, подключённая к этому контакту, показывает её как ложь. В ST логическом потоке, выражение IF statement или петля кажутся исполняемыми, но на самом деле не выполняются, потому что условное выражение отличается в проекте и в контроллере.
3	Выберите подходящую опцию и нажмите **OK**.

ПРИМЕЧАНИЕ: Смотрите Руководство пользователя Вашего контроллера для важной связанной с безопасностью информацией, касающейся загрузки приложений.

Загрузка Вашего приложения в контроллер: Случай 3

Отличающаяся или новая версия приложения уже доступна в контроллере.

Шаг	Действие
1	Для подключения к контроллеру выберите **Online** → **Login to 'Application[YourApplicationName; Plc Logic]'**
2 a	В случае, если контроллер не в режиме RUN и Вы хотите загрузить отличающееся приложение от находящегося в контроллере, появляется следующее сообщение: Обратитесь к сообщениям об опасностях, приведенных ниже, прежде чем нажимать **Yes** для загрузки нового приложения в контроллер, или нажмите **No** для прекращения операции.
2b	В случае если контроллер в режиме RUN и Вы хотите загрузить приложение отличающееся от находящегося в контроллере, появляется следующее сообщение: Обратитесь к сообщениям об опасностях, приведенных ниже, прежде чем нажимать **Yes** для загрузки нового приложения в контроллер, или нажмите **No** для прекращения операции.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННОЕ ДЕЙСТВИЕ ОБОРУДОВАНИЯ

Проверьте, что Вы имеете корректное приложение, прежде чем подтверждать загрузку.

Отказ следовать этим указаниям может привести к смерти, серьёзным травмам или повреждению оборудования.

Если Вы нажмёте **Yes**, работающее приложение в Вашем контроллере, будет перезаписано.

Чтобы помочь предотвратить потерю информации прервите эту операцию нажатием **No** и выполните заранее команду **Source Upload**. Приложение, доступное в данный момент в Вашем контроллере, будет загружено в Ваш ПК. Вы можете затем сравнить его с тем, которое Вы собираетесь загрузить.

Загрузка Вашего приложения в контроллер: Случай 4

Приложение в контроллере ещё не доступно

Шаг	Действие
1	Для подключения к контроллеру выберите **Online** → **Login to 'Application[YourApplicationName; Plc Logic]'**.
2	В случае, когда приложение в контроллере ещё не доступно, Вас попросят подтвердить загрузку. Для этой цели отображается диалоговое окно со следующим текстом: Нажмите **Yes** для загрузки приложения в контроллер, или **No** для прекращения операции.

ПРИМЕЧАНИЕ: Смотрите Руководство пользователя для Вашего контроллера для важной связанной с безопасностью информацией, касающейся загрузки приложений.

Изменения онлайн

Команда **Online Change** изменяет работающую программу приложения и не вызывает процесса перезапуска:

- Программный код может стать другим, чем после полной инициализации, потому что машина сохраняет своё состояние.
- Указатели переменных сохраняют свои значения из последнего цикла. Если есть указатель на переменную, которая изменяет свой размер, из-за онлайн изменений, значение не будет больше верным. Проверьте, что переменные указателей переназначаются в каждом цикле.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННОЕ ДЕЙСТВИЕ ОБОРУДОВАНИЯ

Полностью проверьте код вашего приложения на правильное функционирование, прежде перевода вашей системы в эксплуатацию.

Отказ следовать этим указаниям может привести к смерти, серьёзным травмам или повреждению оборудования.

ПРИМЕЧАНИЕ: За специальной информацией обращайтесь к главе ***Controller States Description*** в Руководстве пользователя Вашего контроллера.

Если проект приложения, работающий в данный момент в контроллере, был изменён после последней загрузки, только изменённые объекты проекта будут загружены в контроллер, в то время как программа продолжает выполняться.

Неявное онлайн изменение

Когда Вы пытаетесь войти снова с изменённым приложением (проверяемым через COMPILEINFO, который был сохранен в папке проекта во время последней загрузки) Вас спросят, хотите ли вы сделать онлайн изменения, загрузку или логин без изменений.

Диалоговое окно логина:

Описание элементов:

Элемент	Описание
Login with online change	Эта опция выбрана по умолчанию. Если Вы подтвердите диалоговое окно нажатием **OK**, модификации будут загружены и немедленно отображены в онлайн виде (мониторинг) соответствующего объекта или объектов.
Login with download	Активируйте эту опцию для загрузки и инициализации проекта приложения полностью.
Login without any change	Активируйте эту опцию с целью сохранить программу, работающую в контроллере, неизменной. Позже может быть выполнена явная загрузка, следовательно, загрузка полного проекта приложения. Также возможно, что Вас спросят снова о желании онлайн изменений, выполненных при следующем релогине.
Details	Нажмите эту кнопку для получения диалогового окна **Application Information** (**Project name, Last modification, IDE version, Author, Description**) для текущего приложения внутри IDE (Integrated Development Environment, т.е., SoMachine) в сравнении с тем, что в данный момент доступен в контроллере. Обращайтесь к следующему изображению и главе ***Информация о приложении*** (*см. стр. 227*).

Диалоговое окно Application Information

За дальнейшей информацией обращайтесь к главе *Логин* *(см. стр. 228)*.

Если онлайн изменения будут вызывать значительные изменения в загружаемом коде, как в примере возможного перемещения адресов указателей или необходимости перенаправления ссылок интерфейсов *(см. стр. 163)* другое диалоговое окно отображается после Вашего подтверждения диалогового окна **Online change** нажатием **OK,** перед тем как загрузка будет выполнена. Оно информирует вас об эффектах, которые вы должны рассмотреть и произвести опцию для отказа от операции онлайн изменений.

ПРИМЕЧАНИЕ: У SoMachine V4.0 и более поздних версий после удаления функций неявной проверки (таких как CheckBounds) из вашего приложения, **Online Change** невозможна, только полная загрузка (download). Будет появляться соответствующее сообщение.

Нажмите кнопку **Details** в этом окне сообщений, чтобы отобразить детальную информацию, такую как число и перечень изменённых интерфейсов, POUs, воздействующих переменных и т.п.

Диалоговое окно Detailed Online Change Information

Явные онлайн изменения

Выполните команды **Online Change** (по умолчанию в **Online** меню), чтобы явно выполнить операцию по изменению онлайн в конкретном приложении.

Online Change изменённого проекта больше невозможна после операции очистки **Clean** (**Build** → **Clean all**, **Build** → **Clean**). В этом случае информация о тех объектах, которые были изменены после последней загрузки, будет удалена. Следовательно, может быть загружен только целый проект.

ПРИМЕЧАНИЕ:

Рассмотрите следующее прежде выполнения команды **Online Change**:

- Проверьте, что изменённый код свободен от логических ошибок.
- Переменные указателей сохраняют свои значения из последнего цикла. Если Вы укажете на переменную, которая была перемещена, значение больше не будет верным. По этой причине переназначайте переменные указателей в каждом цикле.

Информация о процессе загрузки

Когда проект загружается в контроллер полностью при **Login** или частично при **Online Change**, тогда вид **Messages** будет показывать информацию об общем размере кода, размере глобальных данных, требуемом размере памяти в контроллере и, в случае онлайн изменений, также затрагиваемые POUs.

ПРИМЕЧАНИЕ: В онлайн режиме невозможно изменять уставки устройств или модулей. Чтобы изменить параметры устройств, приложение должно быть отсоединено (logged out). В зависимости от системы шин могут быть некоторые специальные параметры, которые разрешено изменять в онлайн режиме.

Загрузка приложения (Загрузка проекта)

При каждой успешной загрузке активное приложение автоматически сохраняется в файле *application.app* в системной папке контроллера, следовательно, делая его доступным как загрузочное приложение. Загрузочное приложение – это проект, который запускается автоматически при запуске контроллера. Чтобы сделать загруженное активное приложение загрузочным, Вы должны выполнить команду **Create boot application** (доступна в **Online** меню).

Команда **Create boot application** будет копировать файл *application.app* в файл, называемый *<projectname>.app*, таким образом делая его загрузочным приложением для контроллера. Вы можете также создать загрузочное приложение в офлайн режиме *(см. стр. 232)*.

Если Вы хотите соединить тот же самый контроллер с системой программирования на другом ПК или восстановить активное приложение через другой ПК без необходимости онлайн изменений (Online Change) или загрузки (download), следуйте шагам, описанным в параграфе *Передача проектов в другие системы.*

Передача проектов в другие системы

Для передачи проектов в другой компьютер используйте архив проекта *(см. SoMachine, Menu Commands, Online Help)*.

Вы можете передать проект, который уже работает на контроллере *xy* из системы программирования на ПК1, в ПК2. Чтобы быть способным пересоединиться от ПК2 к тому же контроллеру *xy* без необходимости онлайн изменений или загрузки, проверьте следующие уставки проекта перед созданием архива проекта.

Выполните следующие шаги:

1. Проверьте, что только библиотеки с определяемыми версиями включены в проект, за исключением чисто интерфейсных библиотек. (Откройте **Library Manager** и проверьте записи со звёздочкой (*) вместо фиксированной версии *(см. SoMachine, Functions and Libraries User Guide)*.)
2. Гарантируйте, что определённая версия компилятора установлена в диалоговом окне **Project Settings** → **Compile options** *(see SoMachine, Menu Commands, Online Help)*.
3. Будьте уверены, что определённый профиль визуализации установлен в диалоговом окне **Project Settings** → **Visualization Profile** (для большей информации обращайтесь к части *Visualization* онлайн справки SoMachine).

4. Проверьте, что открытое в данный момент приложение является таким же, как уже доступное в контроллере. То есть, загруженный проект (обращайтесь к команде **Online** → **Create boot application** *(см. SoMachine, Menu Commands, Online Help)*) должен быть идентичным проекту в системе программирования. Если есть звёздочка после заголовка проекта в поле заголовка окна системы программирования, проект был изменён, но ещё не сохранён. В этом случае он может отличаться от загрузочного проекта. Если необходимо, перед передачей проекта в другой ПК создайте (новый) загрузочный проект (bootproject) – для некоторых контроллеров это делается автоматически при загрузке – и затем загрузите и запустите проект в контроллере.
5. Создайте архив проекта через SoMachine Central. Выберите следующую информацию: **Download information files**, **Library profile**, **Referenced devices**, **Referenced libraries**, **Visualization Profile**.
6. Выйдите из системы. Если необходимо, остановите и перезапустите контроллер *xy* перед пересоединением к ПК2.
7. Извлеките архив проекта на ПК2 с теми же опциями информации, активированными на шаге 5.

Раздел 9.3

Выполнение приложений

Выполнение приложений

Введение

Этот раздел показывает, как запускать / останавливать приложение.

RUN/STOP с SoMachine

Контроллер может запускаться и останавливаться, используя запуск SoMachine на ПК, подключённом к контроллеру.

Нажмите кнопку **Online** → **Start 'Application [ApplicationName: Plc logic]',** или CTRL + F5, или **Start 'Application [ApplicationName: Plc logic]'** в поле меню для запуска приложения.

Нажмите кнопку **Online** → **Stop 'Application [ApplicationName: Plc logic]'** или CTRL +SHIFT + F5, или **Stop 'Application [ApplicationName: Plc logic]'** в поле меню для остановки приложения.

Вход RUN/STOP для контроллеров

Некоторые контроллеры позволяют Вам сконфигурировать вход Run/Stop для управления запуском/остановкой приложения.

Статус	Описание
0	Приложение остановлено. Команда RUN в SoMachine не возможна.
Нарастающий фронт	Запуск приложения.
1	Приложение работает. Команда RUN/STOP в SoMachine возможна.

Обращайтесь к руководству Вашего контроллера, чтобы найти, поддерживает ли он эту функцию.

Конфигурируйте и используйте вход RUN/STOP, если Вы собираетесь использовать удалённые команды для запуска и остановки контроллера. Это наилучший метод помочь гарантировать локальное управление работой контроллера и помочь предотвратить непреднамеренный запуск контроллера из удаленного места.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННЫЙ ЗАПУСК МАШИНЫ ИЛИ ПРОЦЕССА

- Проверьте состояние защиты оборудования Вашей машины или процесса, прежде чем подавать питание на вход Run/Stop.
- Используйте вход Run/Stop для помощи предотвращения непреднамеренного запуска из удалённого места.

Отказ следовать этим указаниям может привести к смерти, серьёзным травмам или повреждению оборудования.

Раздел 9.4

Обслуживание приложений

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Мониторинг	244
Отладка	245

Мониторинг

Обзор

В онлайн режиме есть различные возможности отображения текущих значений объектов в контроллере:

- Вы можете видеть значения объектов на экране редактора программы в режиме онлайн. Детали смотрите в описании соответствующего редактора.
- Вы можете видеть значения объектов в онлайн виде редактора деклараций. За деталями обращайтесь к описанию редактора деклараций *(см. стр. 371)*.
- Вы можете видеть объекты непрерывно в списках через команду **Watch**. За деталями обращайтесь к описанию вида watch / редактора списка watch *(см. стр. 416)*. Вы можете вставлять переменную в вид watch выделением её и выполнением команды **Add watchlist** из контекстного меню.
- Вы можете видеть значения через семплинг трассировки: запись и отображение значений переменной из контроллера. За деталями обращайтесь к описанию функциональности объекта трассировки *(см. стр. 450)*.
- Вы можете видеть значения объектов, которые содержатся в рецептах: Определяемый пользователем набор переменных для записи и наблюдения этих переменных в контроллере. Обращайтесь к описанию менеджера рецептов *(см. стр. 431)*.

За информацией о мониторинге свойств, которые вставлены ниже POUs или функциональных блоков, обращайтесь к главе Свойства *(см. стр. 161)*.

За информацией о мониторинге вызовов функций обращайтесь к главе *Мониторинг атрибутов (см. стр. 556)*.

ПРИМЕЧАНИЕ: Если значение не действительно (например, результат вычисления квадратного корня из отрицательного числа), результат может быть отображён как `NaN` (not a number) или `INF` (infinite value) в зависимости от операции, объекта и конкретной контроллерной платформы. Для большей информации смотрите Руководство по программированию Вашего конкретного контроллера.

Отладка

Обзор

Для обнаружения потенциальных ошибок программирования Вы можете использовать функциональность отладки в онлайн режиме. Вы также можете в определённой мере отлаживать Ваше приложение при симуляции. Хотя симуляция аннулирует необходимость подключения к физическим аппаратным средствам, есть ограничения, из-за которых Вам может понадобиться завершить отладку онлайн.

Вы можете устанавливать точки останова в определённых положениях, чтобы форсировать прерывание выполнения. Определённые условия, такие как: какие задачи идентифицированы, и в каких циклах точки останова должны быть эффективны, могут быть установлены для каждой точки останова (условные точки останова). Обращайтесь к описанию *Точек останова* в данной главе.

Функции шагов доступны, что позволяет выполнить программу по управляемым шагам. Обращайтесь к параграфу *Пошаговое выполнение* (*Stepping*) *(см. стр. 246)*.

При каждом прерывании Вы можете проверять текущие значения переменных. Вы можете наблюдать стек вызовов Call Stack *(см. SoMachine, Menu Commands, Online Help)* для текущей позиции шага.

Вы можете активировать функцию **Flow Control** с целью отслеживать выполняемые части программы приложения. В отличие от стандартных функций мониторинга, которые только показывают значение переменной между двумя циклами выполнения, **Flow Control** обеспечивает значение при каждом конкретном шаге выполнения точно во время выполнения.

Точки остановок

Точка останова, которая устанавливается в программе приложения, будет вызывать прерывание во время выполнения программы. Только задача, имеющая хит точки останова (вызываемая в следующей отлаживаемой задаче) будет останавливаться: однако другие будут продолжать выполняться. Возможные позиции точек останова зависят от редактора. В каждом случае есть точка останова в конце POU.

ПРИМЕЧАНИЕ: Вх/Вых, поддерживаемые отлаживаемой задачей, не обновляются при прерывании в точке останова. Это применяется, даже если опция **Update IO while in stop** разрешена во вкладке **PLC settings** редактора устройств *(см. стр. 121)*.

Обращайтесь к главе *Команды точек останова* для описания команд, касающихся точек останова. Диалоговое окно **Breakpoints** *(see SoMachine, Menu Commands, Online Help)* обеспечивает обзор всех точек останова, позволяет Вам добавлять, удалять и изменять точки останова.

Условные точки останова

Прерывание в точке останова может быть сделано зависимым от числа циклов заданной задачи, или от того, какая задача активна в данный момент. Декларированием специальной задачи отладки Вы можете избежать того, что каждая задача, разделяющая одну и ту же POU, будет влиять на точку останова (обращайтесь к параграфу *Точки останова и шаги в Applications со многими задачами* *(см. стр. 246)*).

Символы точек останова

Символ	Описание
●	точка останова разрешена
○	точка останова запрещена
	Прерывание на точке останова в режиме онлайн
	Текущая позиция шага индицируется жёлтой стрелкой перед соответствующей линией и жёлтой тенью позади связанной операции

Пошаговое выполнение

Пошаговое выполнение (Stepping) позволяет управляемое выполнение программы приложения для целей отладки. Обычно Вы шагаете от одной инструкции к другой с помощью шага входа в инструкцию, шага через следующую инструкцию или шага выхода из инструкции. Обращайтесь к главе *Breakpoint-Related Commands (см. SoMachine, Menu Commands, Online Help)* за описанием команд пошагового выполнения.

Пример операции Step Into

Начиная с точки останова, Вы можете выполнять каждую одиночную линию пошаговой командой.

Step Into, пример

```
1  ldl();
2  erg 0 :=fbinst.ic
3  IF bvar FALSE THEN
4      ivar1 45 :=23;
5  ELSE
6      ivar1 45 :=45;
```

Точки останова и пошаговое выполнение в приложениях с несколькими задачами

Если точка останова может быть хитом нескольких задач, потому что POU используется в нескольких задачах, только задача, которая достигается первой, будет прерываться. Рассмотрите это в случае пошагового выполнения или если вы продолжаете отладку после прерывания. Может быть случай, что другая задача может быть прервана на следующем хите (цикл возможно ещё не закончен). Если должна быть рассмотрена одна определённая задача (отлаживаемая задача), Вы можете указать это в свойствах условия точки останова (диалоговое окно **Breakpoints** → **New Breakpoint**, вкладка **Condition**).

Часть IV

Логические редакторы

Содержание этой части

Эта часть содержит следующие главы:

Глава	Название главы	Страница
10	Редактор FBD/LD/IL	249
11	Редактор Continuous Function Chart (CFC)	297
12	Редактор Sequential Function Chart (SFC)	317
13	Редактор Structured Text (ST)	347

Глава 10

Редактор FBD/LD/IL

Что находится в этой главе?

Эта глава содержит следующие разделы:

Раздел	Тема	Страница
10.1	Информация о редакторе FBD/LD/IL	250
10.2	Элементы FBD/LD/IL	279
10.3	Элементы LD	294

Раздел 10.1

Информация о редакторе FBD/LD/IL

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Редактор FBD/LD/IL	251
Язык Function Block Diagram (FBD) – диаграмм функциональных блоков	252
Язык Ladder Diagram (LD) – лестничных диаграмм	253
Язык Instruction List (IL) – списка инструкций	254
Модификаторы и операторы в IL	256
Работа в FBD и LD редакторе	258
Работа в IL редакторе	262
Позиции курсора в FBD, LD и IL	268
Меню FBD/LD/IL	272
Редактор FBD/LD/IL в режиме онлайн	273

Редактор FBD/LD/IL

Обзор

Комбинированный редактор доступен для редактирования POUs на языках FBD (диаграмм функциональных блоков *(см. стр. 252)*), LD (лестничных диаграмм *(см. стр. 253)*), и IL (списка инструкций *(см. стр. 254)*).

Следовательно, используется общий набор команд и элементов и производится автоматическая внутренняя конверсия между тремя языками. В офлайн режиме программист может переключать виды (**View**).

Имейте в виду, что есть несколько специальных элементов, которые не могут быть преобразованы и, следовательно, будут отображаться на соответствующем языке. Также есть некоторые конструкции, которые не могут конвертироваться однозначно между IL и FBD и, следовательно, будут нормализованы для конверсии назад в FBD; специально, отрицанием выражений и неявным/явным назначением выходов.

Вы можете определить поведение, вид и меню редактора FBD/LD/IL в диалоговых окнах **Customize** и **Options**. Вы также имеете опции для определения отображения комментариев и адресов.

Редактор открывает двухчастное окно. Когда Вы редактируете объект программирования в FBD/LD/IL, верхняя часть содержит редактор объявлений (деклараций) *(см. стр. 372)*, нижняя часть содержит область кода.

Язык программирования для новых объектов указывается при создании объекта.

За дальнейшей информацией обращайтесь к:
- *Работа в виде редактора FBD и LD (см. стр. 258)*
- *Работа в виде редактора IL (см. стр. 262)*

Язык Function Block Diagram (FBD) - диаграмм функциональных блоков

Обзор

Диаграммы функциональных блоков – графически ориентированный язык программирования. Он работает со списком цепей. Каждая Цепь содержит графические структуры блоков и соединительные линии, которые представляют или логические или арифметические выражения, вызов функционального блока, инструкции перехода или возврата.

FBD цепи

Язык лестничных диаграмм (LD)

Обзор

Ladder Diagram (Лестничная диаграмма) – графически ориентированный язык программирования, который напоминает структуру электрической цепи.

С одной стороны, лестничная диаграмма подходит для конструирования логических переключений, с другой стороны, он также позволяет Вам создавать сети как в FBD. Следовательно, LD полезен для управления вызовами других POUs.

Лестничная диаграмма состоит из серии цепей, каждая ограниченная вертикальной линией (шина питания) слева. Цепь содержит диаграмму схемы, сделанной из контактов, катушек, опционно дополнительных POUs (блоков) и соединительных линий.

С левой стороны есть 1 или серия контактов, передающих слева направо условие ON (Вкл) или OFF (Откл), которое соответствует логическим значениям TRUE (Истина) и FALSE (Ложь). Для каждого контакта назначается логическая переменная. Если это значение TRUE, условие будет передано слева направо по соединительной линии. Иначе будет передаваться OFF. Следовательно, катушка или катушки, которые размещены в правой части цепи, получают ON или OFF слева. Соответственно значение TRUE или FALSE будет записано в назначенную логическую переменную.

Цепь Ladder Diagram

Язык Instruction List (IL) – Списка инструкций

Обзор

Instruction List (IL) – похожий на ассемблер, конформный с IEC 61131-3, язык программирования.

Этот язык поддерживает программирование, основанное на аккумуляторе. Операторы IEC 61131-3 поддерживаются также как множественные входы / множественные выходы, отрицания, комментарии, установка / сброс выходов и безусловные / условные переходы.

Каждая инструкция первично основана на загрузке значений в аккумулятор с использованием LD оператора. После этого операция выполняется с первым параметром, взятым из аккумулятора. Результат операции доступен в аккумуляторе, из которого Вы должны сохранить его инструкцией ST.

С целью условного выполнения программы или организации циклов IL поддерживает как операторы сравнения, такие как EQ, GT, LT, GE, LE, NE, и переходы. Последние могут быть безусловными (JMP) или условными (JMPC / JMPCN). Для условных переходов значение аккумулятора сопоставляется с TRUE или FALSE.

Синтаксис

Список инструкций (IL) состоит из серии инструкций. Каждая инструкция начинается в новой линии и содержит оператор и, в зависимости от типа операции, один или больше операндов, разделённых запятыми. Вы можете расширять оператор с помощью модификатора.

В линии перед инструкцией может быть идентификационный знак (метка), сопровождаемый двоеточием (:) (`ml:` в примере, показанном ниже). Метка может быть целью инструкции перехода (`JMPC m1` в примере, показанном ниже).

Помещайте комментарий как последний элемент линии.

Вы можете вставлять пустые линии между инструкциями.

Пример

```
PROGRAM IL VAR
      inst1: TON;
      dwVar: DWORD;
      dwRes: DWORD;
      t1:     TIME;
      tout1: TIME;
      inst2: TON;
      bVar:   BOOL;
END_VAR
LD        bVar                                    variable
ST        inst1.IN            starts timer   with risin...

JMPC      ml
```

```
CAL       inst1( PT:=t1,
          ET:=>tout1)
LD        inst1.Q                is TRUE, PT seconds aft...
ST        inst2.IN               starts timer with risin...
ml:
LD        dwVar
ADD       230
ST        dwRes
```

За дальнейшей информацией обращайтесь к:

- *Работа в виде редактора IL* (*см. стр. 262*)
- *Модификаторы и операторы в IL* (*см. стр. 256*)

Модификаторы и операторы в IL

Модификаторы

Вы можете использовать следующие модификаторы в Instruction List *(см. стр. 254)*.

C	с JMP, CAL, RET:	Инструкция будет выполняться только при условии, что результат предшествующего выражения TRUE.
N	с JMPC, CALC, RETC:	Инструкция будет выполняться только при условии, что результат предшествующего выражения FALSE.
N	другие операторы согласно таблице ***Operators*** ниже (***N*** в колонке ***Modifiers***)	Отрицание операнда (не аккумулятора).

ПРИМЕЧАНИЕ: Обычно не является хорошей практикой использовать выражение CALC (/RETC/JMPC) непосредственно после оператора STN, S или R, так как эти инструкции произвольно изменяют значение аккумулятора и, следовательно, могут привести к трудно обнаруживаемым ошибкам программирования.

Операторы

Таблица показывает какие операторы могут быть использованы в комбинации с указанными модификаторами.

Аккумулятор сохраняет текущее значение, полученное от предыдущей операции.

Оператор	Модификаторы	Значение	Пример
LD	N	Загружает (инвертированное) значение операнда в аккумулятор.	`LD iVar`
ST	N	Сохраняет (инвертированное) содержимое аккумулятора в переменной операнда.	`ST iErg`
S	–	Устанавливает операнд (типа BOOL) в TRUE, когда содержимое аккумулятора TRUE	`S bVar1`
R	–	Устанавливает операнд (типа BOOL) в FALSE, когда содержимое аккумулятора TRUE.	`R bVar1`
AND	N,(	Поразрядное AND аккумулятора и (инвертированного) операнда.	`AND bVar2`
OR	N,(	Поразрядное OR аккумулятора и (инвертированного) операнда.	`OR xVar`
XOR	N,(	Поразрядное исключающее OR аккумулятора и (инвертированного) операнда.	`XOR N, (bVar 1,bVar2)`
NOT	–	Поразрядное отрицание содержимого аккумулятора.	–
ADD	(	Сложение аккумулятора и операнда, результат копируется в аккумулятор.	`ADD(iVar1 ,iVar2)`
SUB	(	Вычитание операнда из аккумулятора, результат копируется в аккумулятор.	`SUB iVar2`
MUL	(	Умножение аккумулятора и операнда, результат копируется в аккумулятор.	`MUL iVar2`

Оператор	Модификаторы	Значение	Пример
DIV	(	Деление аккумулятора и операнда, результат копируется в аккумулятор.	`DIV 44`
GT	(	Проверяет - если аккумулятор больше операнда, результат (BOOL) копируется а аккумулятор; >	`GT 23`
GE	(	Проверяет - если аккумулятор больше или равен операнду, результат (BOOL) копируется в аккумулятор; >=	`GE iVar2`
EQ	(	Проверяет - если аккумулятор равен операнду, результат (BOOL) копируется в аккумулятор; =	`EQ iVar2`
NE	(	Проверяет - если аккумулятор не равен операнду, результат (BOOL) копируется в аккумулятор; <>	`NE iVar1`
LE	(	Проверяет - если аккумулятор меньше или равен операнду, результат (BOOL) копируется в аккумулятор; <=	`LE 5`
LT	(	Проверяет - если аккумулятор меньше операнда, результат (BOOL) копируется в аккумулятор; <	`LT cVar1`
JMP	CN	Безусловный (условный) переход на метку	`JMPN next`
CAL	CN	(Условный) вызов PROGRAM или FUNCTION_BLOCK (если аккумулятор TRUE).	`CAL prog1`
RET	–	Ранний возврат из POU и переход назад к вызывающей POU	`RET`
RET	C	Условный – если аккумулятор TRUE, ранний возврат из POU и переход назад к вызывающей POU	`RETC`
RET	CN	Условный – если аккумулятор FALSE, ранний возврат из POU и переход назад к вызывающей POU	`RETCN`
)	–	Вычислять (завершать) различные операции	–

Смотрите также IEC операторы *(см. стр. 617)* и Работу в редакторе IL *(см. стр. 262)* о том, как использовать и обслуживать многочисленные операнды, комплексные операнды, функцию / метод / функциональный блок / программу / действия вызовов и переходов.

Пример

Пример IL программы, использующей модификации:

```
LD        TRUE            load TRUE to accumulator
ANDN      bVar1           execute AND with negative value of bVar1
JMPC      m1             if accum. is TRUE, jump to label "m1"
LDN       bVar2          store negated value of bVar2...
ST        bRes           ... in bRes
m1:
LD        bVar2          store value of bVar2...
ST        bRes           ... in bRes
```

Работа в редакторе FBD и LD

Обзор

Цепи являются базовыми объектами FBD и LD программирования. Каждая цепь содержит структуру, которая отображает логическое или арифметическое выражение, POU (функцию, программу, вызов функционального блока и т.п.), переход, инструкцию возврата.

Когда Вы создаёте новый объект, окно редактора автоматически содержит 1 пустую цепь.

Обращайтесь к общим уставкам редактора в диалоговом окне **Options**, категория **FBD/LD/IL** за потенциальными опциями дисплея редактора.

Подсказки

Подсказки содержат информацию о параметрах переменных или блоков.

Курсор, помещённый на имя переменной или параметра блока, будет вызывать подсказку. Она показывает соответствующий тип. В случае экземпляров функциональных блоков будет отображаться контекст, имя, тип данных, начальное значение и комментарий. Для IEC операторов `SEL`, `LIMIT`, и `MUX` будет отображаться короткое описание входов. Если это определено, будет отображаться адрес и комментарий символа, также как комментарий операнда (в кавычках во второй линии).

Пример: Подсказка на выход POU

Вставка и расположение элементов

- Команды для работы в редакторе по умолчанию доступны в FBD/LD/IL меню *(см. стр. 272)*. Часто используемые команды доступны также в контекстном меню. Это зависит от текущей позиции курсора или текущего выделения (множественное выделение возможно, смотрите ниже Выделение *(см. стр. 260)*), какой элемент будет вставлен через команду меню.

Вы также можете перетаскивать элементы из меню с помощью мыши из панели инструментов *(см. стр. 280)* в окно редактора или из одной позиции внутри редактора в другую (drag and drop). Для этой цели выберите элемент для перетаскивания нажатием кнопки мыши, при нажатой кнопке переместите элемент в соответствующую цепь вида редактора. Как только Вы достигнете цепи, все возможные позиции для вставки соответствующего типа элемента будут указаны маркерами серых позиций. Когда Вы переместите курсор мыши в одну из этих позиций, маркер изменится на зеленый, и Вы можете отпустить кнопку, чтобы вставить элемент в эту позицию.

Позиции вставки в редакторе LD

- Вы можете использовать команды вырезки, копирования, вставки и удаления, доступные в меню **Edit** для расположения элементов. Вы также можете копировать элемент с помощью перетаскивания: выберите элемент в цепи нажатием кнопки мыши, нажмите клавишу CTRL и при сохранении нажатой кнопки мыши и клавиши переместите элемент в желаемую позицию. При её достижении (зелёный маркер позиции), символ плюса будет добавлен к символу курсора. Тогда отпустите кнопку мыши, чтобы вставить элемент.
- О возможных позициях курсора обращайтесь к *Позициям курсора в FBD, LD, и IL* (*см. стр. 268*).
- Вставка EN/ENO для блоков поддерживается разным образом в FBD и LD редакторах.

Обращайтесь к описанию команды Insert Box за дальнейшей информацией (вставка EN/ENO блоков не поддерживается в редакторе IL).

Навигация в редакторе

- Вы можете использовать клавиши стрелок (ARROW) для перемещения к следующей или предыдущей позиции курсора (*см. стр. 268*). Это также возможно между цепями. Навигация клавишами ← и → следует потоку сигнала, нормальное направление которого слева направо, или наоборот. В случае прерывания линии, следующая позиция курсора может также быть левее текущей отмеченной позиции. Если Вы нажимаете клавиши ↑ или ↓, выделение перемещается в верхнюю или нижнюю соседние текущие позиции, если эти соседи есть в той же логической группе (например, вывод или блок). Если такой группы не существует, выделение переходит на ближайший соседний элемент выше или ниже. Навигация через параллельно соединённые элементы производится между первыми ветвлениями.
- Нажмите клавишу HOME для перемещения к первому элементу. Нажмите клавишу END для перемещения к последнему элементу цепи.
- Используйте клавишу TAB для перехода к следующей или предыдущей позиции курсора (*см. стр. 268*) внутри цепи.
- Нажмите CTRL + HOME для перемещения к началу документа и отметка первой цепи.
- Нажмите CTRL + END для перемещения к концу документа и отметка последней цепи.
- Нажмите PAGE UP для перемещения вверх на 1 экран и отметка самого верхнего прямоугольника.
- Нажмите PAGE DOWN для перемещения вниз на 1 экран и отметка самого верхнего прямоугольника.

Выделение

- Вы можете выбрать элемент, а также цепь, через размещение соответствующей позиции курсора с помощью нажатия мышью или использования клавиш стрелок или табуляции. Выделенные элементы индицируются как оттенённые красным. Также обращайтесь к *Позициям курсора в FBD, LD и IL* *(см. стр. 268)*.
- В LD редакторе Вы также можете выделять линии между элементами в порядке выполнения команд, например, для вставки дальнейшего элемента в этой позиции.

Выделение линии в LD редакторе

- Выделение нескольких элементов или цепей возможно с помощью поддержания нажатой клавиши CTRL, когда выделяются желаемые элементы один за другим.

Открытие функционального блока

Если функциональный блок вставлен в редактор, вы можете открыть этот блок двойным нажатием кнопки мыши.

Альтернативно Вы можете использовать команду **Browse - Go To Definition** из контекстного меню.

Редактор LD

Для получения информации о языках обращайтесь к:

- *Function Block Diagram - FBD (см. стр. 252)*
- *Ladder Diagram - LD (см. стр. 253)*

Работа в IL редакторе

Обзор

Редактор списка инструкций IL Instruction List *(см. стр. 254)* является табличным редактором. Структура цепи FBD или LD программ также представляется в IL программе. Обычно одной цепи *(см. стр. 282)* достаточно в LD программе, но, рассматривая переключение между FBD, LD и IL, Вы также можете использовать цепи для структурирования LD программы.

Также обращайтесь к основным уставкам редактора в диалоговом окне **Options**, категория **FBD/IL/LD**.

Подсказки

Подсказки содержат информацию о параметрах переменной или блока. Обращайтесь к *Работе в виде редактора FBD и LD (см. стр. 258)*.

Вставка и расположение элементов

- Команды для работы в редакторе доступны в **FBD/LD/IL** меню. Часто используемые команды также доступны в контекстном меню.
- Программные единицы, которые являются элементами, вставляются каждый в текущей позиции курсора через команды **Insert**, доступные в **FBD/LD/IL** меню.
- Вы можете использовать команды вырезки, копирования, вставки и удаления, доступные в меню **Edit** для расположения элементов.
- Смотрите также некоторую информацию по программированию в Instruction List - IL *(см. стр. 254)*.
- Операторы с EN/ENO функциональностью могут быть вставлены только внутри FBD и LD редактора.

Эта глава объясняет, как табличный редактор структурирован, как Вы можете осуществлять навигацию в редакторе и как использовать комплексные операнды, вызовы и переходы

.

Структура IL табличного редактора

IL табличный редактор

Каждая линия программы, записанная в строке таблицы, структурирована в полях по колонкам таблицы:

Колонка	Содержание...	Описание
1	оператор	Это поле содержит IL оператор (LD, ST, CAL, AND, OR, и т.п.) или имя функции. В случае вызова функционального блока соответственные параметры также специфицируются здесь. Вводите поля префиксов `:=` или `=>`. За дальнейшей информацией обращайтесь к ***Модификаторам и операторам в IL*** *(см. стр. 256)*.
2	операнд	Это поле содержит точно один операнд или метку перехода. Если требуется более одного оператора (мночисленные/расширяемые AND A, B, C, или вызовы функций с несколькими параметрами), записывайте их на следующих линиях и оставляйте поле операнда пустым. В этом случае добавляется разделяющая параметры запятая. В случае вызова функционального блока, программы или действия (action) добавляйте соответствующие открывающие и/или закрывающие скобки.
3	адрес	Это поле содержит адрес операнда как определено в декларационной части. Вы не можете редактировать это поле. Используйте опцию **Show symbol address**, чтобы переключать его на Вкл или Откл.
4	комментарий к символу	Это поле содержит комментарий, как определено для операнда в декларационной части. Вы не можете редактировать это поле. Используйте опцию **Show symbol address**, чтобы переключать его на Вкл или Откл.
5	комментарий к операнду	Это поле содержит комментарий к текущей линии. Оно редактируемо и может переключаться между Вкл и Откл через опцию **Show operand comment**.

Навигация в таблице

- Клавиши стрелок UP и DOWN: Перемещение к полю выше или ниже.
- TAB: Перемещение по линии к полю справа.
- SHIFT + TAB: Перемещение по линии к полю слева.
- SPACE: Открывает текущее выбранное поле для редактирования. Альтернативно выполните дальнейшее нажатие мышью на поле. Если применимо, ассистент ввода будет доступен через кнопку …. Вы можете закрыть открытое в данный момент поле редактирования нажатием ENTER, подтверждением текущих изменений или нажатием ESC отмены сделанных изменений.
- CTRL + ENTER: Ввод новой линии ниже текущей.
- DEL: Удаление текущей линии, то есть имеющейся на выделенном в данный момент поле.
- **Cut**, **Copy**, **Paste**: Для копирования одной или нескольких линий выберите, по крайней мере, 1 поле линии или линии и выполните команду **Copy**. Чтобы вырезать линию, используйте команду **Cut**. **Paste** будет вставлять предварительно скопированные / вырезанные линии перед линией, где в данный момент выделено поле. Если нет выбранного поля, они будут вставлены в конце цепи.
- CTRL + HOME перемещает к началу документа и помечает первую цепь.
- CTRL + END перемещает к концу документа и помечает конечную цепь.
- PAGE UP перемещает на 1 экран вверх и помечает самый верхний прямоугольник.
- PAGE DOWN перемещает на 1 экран вниз и помечает самый верхний прямоугольник.

Несколько операндов (Расширяемые операторы)

Если один и тот же оператор *(см. стр. 256)* используется с несколькими операндами, возможно 2 пути программирования:

- Операнды вводятся в последовательных линиях, разделённые запятыми, например:

```
LD      7
ADD     2,
        4,
        7
ST      iVar
```

- Инструкции повторяются в последовательных линиях, например:

```
LD      7
ADD     2
ADD     4
ADD     7
ST      iVar
```

Комплексные операнды

Если должен использоваться комплексный операнд, введите открывающую скобку перед, затем используйте последующие линии для конкретных компонентов операнда. Ниже их в отдельной линии введите закрывающую скобку.

Пример: Вращение строки на 1 символ при каждом цикле.

Соответствующий ST код:

```
stRotate := CONCAT(RIGHT(stRotate, (LEN(stRotate) - 1)),
 (LEFT(stRotate, 1)));
```

```
LD        stRotate
RIGHT(    stRotate
LEN
SUB       1
)
CONCAT(   stRotate
LEFT      1
)
ST        stRotate
```

Вызовы функций

Введите имя функции в поле оператора. Возьмите (первый) входной параметр как операнд в предшествующей LD операции. Если есть последующие параметры, берите следующие параметры в той же линии, что и имя функции. Вы можете добавить дальнейшие параметры в этой линии, разделённые запятыми, или в последующих линиях.

Возвращаемое функцией значение будет сохранено в аккумуляторе. Применяется следующее ограничение относительно IEC стандарта.

ПРИМЕЧАНИЕ: Вызов функции с множеством возвращаемых значений невозможен. Только одно возвращаемое значение может использоваться для последующей операции.

Пример: Вызывается функция `GeomAverage`, которая имеет 3 входных параметра. Первый задаётся `X7` в предшествующей операции. Следующий, `25`, задан с именем функции. Третий задаётся переменной `tvar`, или в той же линии или в последовательной. Возвращаемое значение назначается переменной `Ave`.

Соответствующий ST код:

```
Ave := GeomAverage(X7, 25, tvar);
```

Вызов функции в IL:

```
LD             X7
GeomAverage    25
                     tvar
ST             Ave
```

Вызовы функциональных блоков и вызовы программ

Используйте оператор CAL- или CALC *(см. стр. 256)*. Введите имя экземпляра функционального блока или имя программы в поле операнда (вторая колонка), сопровождаемые открывающей скобкой. Введите входные параметры, каждый в одной из следующих линий:
Поле оператора: Имя параметра
Поле префикса:

- `:=` для входных параметров
- `=>` для выходных параметров

Поле операнда: Текущий параметр
Поле постфикса:

- `,` если есть последующий параметр
  `)` после последнего параметра
- `()` в вызовах без параметров

Пример: Вызов `POUToCAll` 2 входными и 2 выходными параметрами.

Соответствующий ST код:

```
POUToCall(Counter := iCounter, iDecrement:=2, bError=>bErr, wError=>wResult);
```

Вызов программы в IL с входными и выходными параметрами:

```
1  PROGRAM IL_EXAMPLE
2  VAR
3      bErr: BOOL;
4      wResult: WORD;
5  END_VAR
6
```

```
1
   CAL          POUToCall(
       Counter := iCounter
    iDecrement:= 2
        wError=> wResult)
   LD           POUToCall.bError
   ST           bErr
```

Нет необходимости использовать все параметры функционального блока или программы.

ПРИМЕЧАНИЕ: Комплексные выражения использоваться не могут. Они должны быть назначены на вход функционального блока или программы перед вызовом инструкции.

Вызов действия (Action)

Выполняемое как вызов функционального блока или программы, имя действия добавляется к имени экземпляра или имени программы.

Пример: Вызов действия `ResetAction`.

Соответствующий ST код:

```
Inst.ResetAction();
```

Вызов действия в IL:

```
CAL      Inst.ResetAction()
```

Вызов метода

Выполняемое как вызов функции, имя экземпляра с добавленным именем метода вводится в первой колонке (оператор).

Пример: Call of method `Home`.

Соответствующий ST код:

```
Z := IHome.Home(TRUE, TRUE, TRUE);
```

Вызов метода в IL:

```
LD              TRUE
IHome.Home      TRUE
                TRUE
ST              Z
```

Переход

Переход *(см. стр. 284)* программируется `JMP` в первой колонке (оператор) и имя метки во второй колонке (операнд). Метка определена в целевой цепи в поле метки *(см. стр. 284)*.

Список операторов, предшествующих безусловному переходу, должен заканчиваться одной из следующих команд: ST, STN, S, R, CAL, RET или другим JMP.

Это не случай для условного перехода *(см. стр. 284)*. Выполнение перехода зависит от загруженного значения.

Пример: Инструкция условного перехода; в случае `bCallRestAction` равно TRUE, программа должна перейти к цепи, отмеченной меткой `Cont`.

Инструкция условного перехода в IL:

```
LDN  bCallResetAction
JMPC Cont
```

Позиции курсора в FBD, LD и IL

Редактор IL

Это текстовый редактор, структурированный в форме таблицы. Каждая ячейка таблицы является возможной позицией курсора. Обращайтесь также к *Работе с видом редактора IL (см. стр. 262)*.

Редакторы FBD и LD

Это графические редакторы, смотрите ниже примеры с (1) по (15), показывающие возможные позиции курсора: текст, вход, выход, блок, контакт, катушка, возврат, переход, линия между элементами и цепь.

Такие действия как вырезание, копирование, вставка, удаление и другие специфические для редактора команды относятся к текущей позиции курсора или выбранного элемента. Смотрите *Работа с видом редактора IL (см. стр. 262)*.

Обычно в FBD пунктирный прямоугольник вокруг соответствующего элемента указывает текущую позицию курсора. Дополнительно тексты и блоки становятся затенёнными голубым или красным.

В LD катушки и контакты становятся выделенными красным, как только курсор позиционируется на них.

Позиция курсора определяет какие элементы доступны в контекстном меню для выбора вставки *(см. стр. 258)*.

Возможные позиции курсора

(1) Каждое текстовое поле

На левом изображении возможные позиции курсора отмечены красным. Правое изображение показывает блок с курсором, помещённым в поле `AND`. Примите во внимание возможность ввода адресов вместо имён переменных, что конфигурируется соответственно в диалоговом окне **FBD, LD and IL editor Options**.

(2) Каждый вход:

(3) Каждый оператор, функция или функциональный блок:

(4) Выходы, если назначение или переход следуют дальше:

(5) Непосредственно перед линией пересечения выше назначения перед инструкцией перехода или возврата:

(6) Выделенная крайняя правая позиция курсора в цепи или в любом другом месте в цепи между другими позициями. Это будет выделять всю цепь:

(7) Линия пересечения непосредственно перед назначением:

(8) Каждый контакт:

(9) Каждая катушка:

(10) Каждый возврат и переход:

(11) Линия соединения между контактами и катушками:

(12) Ответвление или дополнительная цепь внутри цепи:

(13) Линия соединения между параллельными контактами (Поз. 1...4):

(14) Перед или после цепи:

Вы можете добавлять новые цепи в крайней левой части редактора. Вставка новой цепи перед существующей цепью, возможно также перед цепью 1.

(15) Начало или конец цепи:

Вы можете добавлять контакты и функциональные блоки в начале цепи в поле **Start here**. Вы можете добавлять элементы возврата, переходы и катушки в конце цепи в поле **Add output or jump here**.

Меню FBD/LD/IL

Обзор

Когда курсор помещается в окно редактора FBD/LD/IL *(см. стр. 251)*, в поле меню доступно меню FBD/LD/IL, обеспечивая команды для выбранного в данный момент вида редактора.

FBD/LD/IL меню в виде редактора FBD:

- За описанием команд обращайтесь к главе *Команды FBD/LD/IL редактора*.
- За конфигурацией меню обращайтесь к описанию Настройки меню (**Customize Menu)**.

Редактор FBD/LD/IL в режиме онлайн

Обзор

В онлайн режиме редактор FBD/LD/IL обеспечивает виды для мониторинга *(см. стр. 273)* и для записи и форсирования значений и выражений в контроллере.

Доступна функциональность отладки (точки останова, пошаговое выполнение и т.п.), *см. Точки останова или позиции прерываний (см. стр. 277)*.

- Для информации как открывать объекты в онлайн режиме обращайтесь к главе *Интерфейс пользователя в онлайн режиме (см. стр. 49)*.
- Имейте виду, что окно редактора FBD, LD, или IL объекта также содержит редактор декларации **Declaration Editor** в верхней части. Также обращайтесь к главе Редактор декларации в онлайн режиме *(см. стр. 376)*.

Мониторинг

Если мониторинг в линиях явно не деактивирован в диалоговом окне **Options**, он будет дополнен в FBD или LD редакторе маленькими окнами мониторинга у каждой переменной или дополнительной колонкой мониторинга, показывающей текущие значения (линейный мониторинг). Это даже в случае неназначенных входов и выходов функционального блока.

Окно линейного мониторинга переменной показывает маленький красный треугольник в левом верхнем углу, если переменная в данный момент форсирована *(см. стр. 276)*, маленький голубой треугольник в левом нижнем углу, если переменная подготовлена в данный момент к форсированию или записи. В LD для контактов и катушек текущее подготовленное значение (TRUE или FALSE) будет отображаться справа внизу элемента.

Пример для переменной, которая в данный момент форсирована и подготовлена для реализации форсирования

Пример для контакта переменной, которая подготовлена в данный момент для записи или форсирования значением TRUE

Вид онлайн FBD программы

SP_for_Win32.App.CallTest

Expression	Comment	Type	Value	Prepared value
Inst2		UpAndDown		
Enable		BOOL	TRUE	
Amplitude		INT	200	
GoHome		BOOL	FALSE	
Value		INT	41	
Up		BOOL	FALSE	
Down		BOOL	TRUE	
Counter		DINT	2159	
diValue		DINT	41	
X		BOOL	FALSE	
X		INT	0	
X2		INT	0	
Up		BOOL	TRUE	
Down		BOOL	FALSE	
MyCounter	Vars for third network	DINT	F 40	<Unforce and restore>
MyDownCounter		DINT	-2	
Err		BOOL	FALSE	
ErrCode		WORD	0	
stRotate	Vars for fifth network	STRING	'pcom'	'abc'
Ave	Vars for fifth network	DINT	38	

Онлайн IL программы

В виде онлайн ступенчатые цепи имеют анимированные соединения:

- Соединения со значением TRUE отображаются жирным синим цветом.
- Соединения со значением FALSE отображаются жирным чёрным цветом.
- Соединения с неизвестным значением, или с аналоговым значением отображаются видом (черным цветом и не жирной линией).

Значения переменных соединений вычисляются из наблюдаемых значений.

Вид онлайн LD программы

Откройте функцию двойным нажатием или выполните команду **Browse - Go To Definition** из контекстного меню. Обращайтесь к описанию *Интерфейса пользователя в онлайн режиме* *(см. стр. 49)* для дальнейшей информации.

Форсирование / запись переменных

В онлайн режиме Вы можете подготовить значение для форсирования или записи переменной в редакторе деклараций *(см. стр. 376)* или внутри редактора. Дважды нажмите переменную в редакторе для открытия следующего диалогового окна:

Диалоговое окно **Prepare Value**

Вы найдёте имя переменной, дополненное её путём в дереве устройств (**Expression**), её тип и текущее значение. Активированием соответствующей позиции Вы можете сделать следующее:

- Подготовить новое значение, которое должно быть введено в поле редактирования.
- Удалить подготовленное значения.
- Реализация форсированной переменной.
- Реализация форсированной переменной и переустановка её на значение, которое было назначено как раз перед форсированием.

Выбранное действие будет выполнено с помощью команды меню **Force values** (в меню **Online**) или нажатием F7.

За информацией о том, как текущее состояние переменной (форсированное, подготовленное значение) индицируется в соответствующем элементе цепи, обращайтесь к секции *Мониторинг (см. стр. 273)*

Точки останова или позиции прерываний

Возможные позиции, которые Вы можете определить как точки останова (позиции прерывания) для целей отладки, - те позиции, в которых значения переменных могут изменяться (операторы), в которых поток программы расширяется или в которой вызывается другая POU.

Это следующие позиции:

- На цепи в целом, таким образом, что контрольная точка будет применена к первой возможной позиции в цепи.
- На блоке *(см. стр. 285)*, если он содержит оператор. Следовательно, это невозможно на блоках оператора таких как, например, `ADD, DIV`. См. примечание ниже.
- На назначении.
- В конце POU в точке возврата к вызывающему; в онлайн режиме автоматически для этой цели будет отображаться пустая цепь. Вместо номера цепи она идентифицируется с помощью `RET`.

ПРИМЕЧАНИЕ: Вы не можете установить точку останова непосредственно на первом блоке цепи. Если, однако, точка останова устанавливается на всей цепи, позиция прерывания будет автоматически приложена к первому блоку.

За текущими возможными позициями обращайтесь к списку выделений внутри диалогового окна **View** → **Breakpoints**.

Цепи, содержащие какие-либо активные позиции точек останова, отмечены символом прерывания (заполненным красным кругом) справа от номера цепи и затенённым красным прямоугольником фона в первой возможной позиции прерывания внутри цепи. Деактивированные позиции точка прерывания индицируется незаполненным красным кругом или окружающим незаполненным красным прямоугольником.

Установленная точка останова и достигнутая точка останова

1 точка останова установлена
2 точка останова достигнута

Как только во время пошагового выполнения или выполнения программы достигнута точка останова, жёлтая стрелка будет отображена в символе точки останова и красная затенённая область будет заменена на жёлтую.

Позиция прерывания, показанная в FBD

Позиция прерывания, показанная в IL

ПРИМЕЧАНИЕ: Точка останова будет установлена автоматически во всех методах, которые могут быть вызваны. Если вызывается управляемый интерфейсом метод, точки останова будут установлены во всех методах функциональных блоков, реализующих этот интерфейс и во всех производных функциональных блоках, которые присоединены к методу

Раздел 10.2

Элементы FBD/LD/IL

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Панель инструментов FBD/LD/IL	280
Цепь FBD/LD/IL	282
Назначение в FBD/LD/IL	284
Переход в FBD/LD/IL	284
Метка в FBD/LD/IL	284
Блоки в FBD/LD/IL	285
Инструкция `RETURN` в FBD/LD/IL	286
Ветвление / Подвешивание катушек FBD/LD/IL	287
Параллельное ветвление	290
Set/Reset (установка/сброс) в FBD/LD/IL	292
Set/Reset катушки	293

Панель инструментов FBD/LD/IL

Обзор

Редактор FBD/LD/IL *(см. стр. 251)* обеспечивает панель инструментов, которая предлагает элементы программирования для вставки в окно редактирования методом перетаскивания. Откройте панель инструментов выполнением команды **ToolBox**, которая находится в меню **View**.

Она зависит от активного в данный момент вида редактора, элементы которого доступны для вставки (см. соответствующее описание команд вставки).

Элементы отсортированы по категориям: **General** (общие элементы, такие как **Network, Assignment** и т.п.), **Boolean operators**, **Math operators**, **Other operators** (например, `SEL, MUX, LIMIT`, и `MOVE`), **Function blocks** (например, `R_TRIG, F_TRIG, RS, SR, TON, TOF, CTD, CTU`), **Ladder elements**, и **POUs** (определённые пользователем).

Категория **POUs** перечисляет все POUs, которые были определены ниже этого же приложения, что и FBD/LD/IL объект, который открыт в редакторе. Если POU назначено битовое изображение в его свойствах, тогда оно будет отображено перед именем POU. Иначе для индикации типа POU будет использоваться стандартная иконка. Список будет обновляться автоматически, когда POUs добавляются или удаляются из приложения.

Категория **Other operators** содержит среди операторов `SEL, MUX, LIMIT`, и `MOVE` элемент преобразования прототипа. Вы можете перетащить этот элемент в соответствующую позицию цепи. Преобразование типа устанавливается автоматически в зависимости от требуемого типа для позиции вставки. Однако в некоторых ситуациях требуемое преобразование типа не может быть определено автоматически. В этом случае измените элемент вручную.

Чтобы развернуть папки категории, нажмите кнопку, показывающую соответствующее название категории. Смотрите в следующем изображении: Категория **General** развёрнута, остальные свёрнуты. Изображение показывает пример вставки элемента **Assignment** перетаскиванием из панели инструментов.

Развёрнута только секция **General** в панели инструментов:

Вставка из панели инструментов

Для вставки элемента в редактор выберите его в панели инструментов нажатием мыши, и принесити его в окно редактора методом перетаскивания (drag and drop). Возможные позиции вставки будут индицироваться маркерами позиции, которые появляются, пока элемент перетаскивается – поддержание нажатым кнопки мыши – через окно редактора. Ближайшая возможная позиция высвечивается зелёным цветом. Когда Вы отпускаете кнопку мыши, элемент будет вставлен в зелёную позицию.

Если Вы перетащите блочный элемент на существующий блочный элемент, новый заменит старый. Если входы и выходы уже были назначены, они останутся назначенными, но они не будут подсоединены к новому блочному элементу.

Цепь в FBD/LD/IL

Обзор

Цепь является основным элементом программы FBD *(см. стр. 252)* или LD *(см. стр. 253)*. В FBD/LD редакторе цепи размещены в вертикальном списке. Каждая цепь назначается на левой стороне номером последовательной цепи и имеет структуру, состоящую или логических или арифметических выражений, программ, вызовов функций или функциональных блоков и, возможно, инструкций переходов и возвратов.

IL редактор *(см. стр. 254)* из-за общей основы редактора с FBD и LD редакторами, также использует элемент цепи. Если объект первоначально был запрограммирован в FBD или LD и затем преобразован в IL, цепи будут всё ещё присутствовать в IL программе. И наоборот, если Вы начнёте программирование объекта в IL, Вам потребуется хотя бы 1 элемент цепи, который мог содержать все инструкции, но Вы можете также использовать последующие цепи для структурирования программы.

Цепь опционно может получить заданные заголовок, комментарий и метку *(см. стр. 284)*.

Вы можете включать и отключать доступность полей заголовков и комментариев в диалоговом окне опций редакторов FBD, LD и IL. Если опция активирована, нажмите цепь непосредственно ниже верхней границы для открытия и редактирования поля заголовка. Для ввода комментария соответственно откройте и редактируйте поле, находящееся ниже поля заголовка. Комментарий может занимать несколько строк. Нажмите ENTER, чтобы вставить разрыв строки. Нажмите CTRL + ENTER, чтобы прекратить ввод текста комментария.

В диалоговом окне опций редакторов FBD, LD и IL задается то, отображается ли и как комментарий в цепях редакторов.

Чтобы задать метку *(см. стр. 284)*, которая затем может быть адресована в переходе *(см. стр. 284)*, используйте команду **Insert label**. Если метка определена, она будет отображаться ниже полей заголовка и комментария или - если эти поля недоступны - непосредственно ниже верхней границы цепи.

Комментарии и метка в цепи

Вы можете установить цепь в состояние комментария. Это указывает, что цепь не выполняется, но отображается и поддерживается как комментарий. Для достижения этого используйте команду **Toggle network comment state**.

В выбранной в данный момент цепи (6 вариант позиции курсора *(см. стр. 268)*) Вы можете применять команды по умолчанию для копирования, вырезания, вставки и удаления.

ПРИМЕЧАНИЕ: Нажатие правой кнопки (6 вариант позиции курсора *(см. стр. 268)*) на заголовках, комментариях или метках будет выделять в этой записи только целиком всю цепь. Поэтому выполнение команд по умолчанию не будет иметь влияния на цепь.

Чтобы вставить цепь, используйте команду **Insert Network** или перетащите её из панели инструментов *(см. стр. 280)*. Цепь со всеми присутствующими элементами также может быть скопирована или перемещена *(см. стр. 258)* способом перетаскивания внутри редактора.

Вы также можете создавать подцепи *(см. стр. 287)* вставкой ветвей.

Цепь RET

В онлайн режиме дополнительная пустая цепь будет отображаться ниже существующих цепей, вместо номера цепи она идентифицируется как RET.

Она представляет позицию, в которой выполнение будет возвращаться к вызывающей POU и обеспечивать возможную позицию прерывания *(см. стр. 273)*.

Назначение в FBD/LD/IL

Обзор

В зависимости от выбранной позиции курсора *(см. стр. 268)* в FBD или LD назначение будет вставляться непосредственно перед выбранным входом (вариант 2 позиции курсора *(см. стр. 268)*), непосредственно после выбранного выхода (вариант 4 позиции курсора *(см. стр. 268)*) или в конце цепи (вариант 6 позиции курсора *(см. стр. 268)*). В цепи LD назначение будет отображаться как катушка *(см. стр. 296)*.

Альтернативно перетаскивайте элемент назначения из панели инструментов *(см. стр. 280)* или копируйте или перемещайте *(см. стр. 258)* его внутри вида редактора.

После вставки текстовая строка ??? может быть замещена именем переменной, которая назначена. Для этого используйте кнопку **…**, чтобы открыть **Input Assistant**.

В IL *(см. стр. 254)* назначение программируется через инструкции LD и ST. Обращайтесь к *Модификаторам и операторам в IL (см. стр. 256)*.

Переход в FBD/LD/IL

Обзор

В зависимости от выбранной позиции курсора *(см. стр. 268)* в FBD *(см. стр. 252)* или LD *(см. стр. 253)* переход будет вставлен непосредственно перед выбранным входом (вариант 2 позиции курсора), непосредственно после выбранного выхода (вариант 4 позиции курсора) или в конце цепи (вариант 6 позиции курсора).

Альтернативно перетаскивайте элемент назначения из панели инструментов *(см. стр. 280)* или копируйте или перемещайте *(см. стр. 258)* его внутри вида редактора.

После вставки Вы можете заменить автоматически вставляемую текстовую строку ??? меткой, на которую должен быть осуществлён переход.

В IL *(см. стр. 254)* переход вставляется через инструкцию JMP. Смотрите в этом контексте описание операторов и модификаторов и в IL *(см. стр. 256)*.

Метки FBD/LD/IL

Обзор

Ниже поля комментария цепи каждая цепь FBD *(см. стр. 252)*, LD *(см. стр. 253)* или IL имеет текстовое поле ввода для задания метки. Метка является дополнительным идентификатором цепи и может адресоваться при определении перехода *(см. стр. 284)*. Она может состоять из любой последовательности символов.

Позиция метки в цепи

Смотрите диалоговое окно **Tools** → **Options** → **FBD, LD and IL editor** для определения отображения комментария и заголовка.

Блоки в FBD/LD/IL

Обзор

Блок, вставляемый в цепь FBD *(см. стр. 252)*, LD *(см. стр. 253)* или IL *(см. стр. 254)*, является комплексным элементом и может представлять дополнительные функции, такие как таймеры, счётчики, арифметические операторы или также программы, IEC функции и IEC функциональные блоки.

Блок может иметь один или более входов и выходов и может поставляться системной библиотекой или может быть запрограммирован. Однако, по крайней мере, 1 вход и 1 выход должны быть назначены на булевы значения.

Он поставляется с соответствующим модулем и, если опция **Show box icon** активирована, иконка будет отображаться внутри блока.

Использование в FBD, LD

Вы можете позиционировать блок в цепи LD или цепи FBD, используя команды **Insert Box**, **Insert Empty Box**. Альтернативно Вы можете вставить его из панели инструментов *(см. стр. 280)* или копировать или перемещать его внутри этого с помощью перетаскивания. За дальнейшей информацией обращайтесь к описанию команды **Insert Box**.

Использование в IL

В IL *(см. стр. 254)* программе, инструкция CAL *(см. стр. 256)* с параметрами будет вставлена с целью представления блокового элемента.

Вы можете обновлять параметры блока (входы и выходы) – в случае модификации интерфейса блока – текущим представлением с помощью команды **Update parameters** без необходимости повторной вставки.

Инструкция RETURN в FBD/LD/IL

Обзор

С инструкцией RETURN FBD *(см. стр. 252)*, LD *(см. стр. 253)* или IL *(см. стр. 254)* POU могут быть выполнены.

В цепи FBD или LD Вы можете поместить её параллельно или в конце предыдущих элементов. Если вход RETURN TRUE, то из выполнения POU будет немедленно осуществлён выход.

Выполните команду **Insert Return**, чтобы вставить инструкцию RETURN. Альтернативно перетащите элемент из панели инструментов *(см. стр. 280)* или скопируйте или переместите *(см. стр. 258)* его из другой позиции внутри редактора.

Элемент RETURN

В IL инструкция RET *(см. стр. 256)* используется для той же самой цели.

Ветвление / Подвешивание катушек в FBD/LD/IL

Обзор

В цепи Function Block Diagram *(см. стр. 252)* или Ladder Diagram *(см. стр. 253)* ветвление (branch) или подвешивание катушек разделяет выполнение линии в текущей позиции курсора. Выполняемая линия будет продолжаться в 2 дополнительных подцепях, которые будут выполняться по одной после каждой другой сверху вниз. Каждая подцепь может иметь дальнейшее ветвление, таким образом позволяя многочисленное ветвление внутри цепи.

Каждая дополнительная цепь получает собственный маркер (символ правого верхнего угла прямоугольника). Вы можете выбрать его (вариант 11 позиции курсора *(см. стр. 268)*) с той целью, чтобы произвести действия в этой ветви ветвления.

Маркеры ветвлений

В FBD вставьте ветвление через команду **Insert branch**. Альтернативно перетащите элемент из панели инструментов *(см. стр. 280)*. О возможных позициях вставки обращайтесь к описанию команды **Insert branch**.

ПРИМЕЧАНИЕ: Вырезание и вставка не осуществлены для дополнительных подцепей.

Ветвление было введено на выходе блока SUB в примере, показанном ниже. Это создаёт 2 подцепи, каждая выделяемая своим маркером подцепи. После этого блоки `ADD` были добавлены в каждую подцепь.

Цепь в FBD с вставленным ветвлением

Чтобы удалить подцепь, вначале удалите все элементы подцепи, то есть все элементы, расположенные правее маркера подцепи. Затем выберите маркер и выполните стандартную команду **Delete**, или нажмите клавишу DEL.

В последующем изображении 3-входовый элемент OR был удалён перед тем, как Вы можете выбрать и удалить маркер нижней подцепи.

Удаление ветвления или подцепи

Выполнение в онлайн режиме

Конкретные ветвления будут выполняться слева направо и затем сверху вниз.

IL (Instruction List)

IL *(см. стр. 254)* не поддерживает цепи с ветвлениями. Они будут оставаться в оригинальном представлении.

Параллельные ветвления

Вы можете использовать параллельные ветвления для оценки реализации параллельных ветвей *(см. стр. 290)* в цепях лестничных диаграмм.

В отличие от открытых ветвлений (без точки объединения) параллельные ветви закрыты. Они имеют точки разделения и объединения.

Параллельные ветви

Обзор

Параллельное ветвление позволяет Вам осуществить параллельное соединение логических элементов. Это достигается через методологию, описанную как Short Circuit Evaluation (SCE). SCE позволяет Вам шунтировать выполнение функционального блока с логическим выходом, если определённые параллельные условия оцениваются как TRUE. Условие может быть представлено в LD редакторе параллельной ветвью к параллельной ветви функционального блока. SCE условие определяется одним или несколькими контактами внутри ветви, соединёнными параллельно или последовательно.

ПРИМЕЧАНИЕ: Термин ветвь также используется для другого элемента, который разделяет передачу сигнала. Это ветвление *(см. стр. 287)* как противоположность параллельному ветвлению не имеет точки объединения.

Параллельное ветвление работает следующим образом: сначала будут проанализированы ветви, не содержащие функциональных блоков. Если одна из этих ветвей оценивается как TRUE, то функциональный блок не будет вызываться и значение на входе ветвления функционального блока будет передано на выход. Если SCE условие оценивается как FALSE, тогда будет вызываться функциональный блок и логический результат вызова функционального блока будет передан на выход.

Если все ветви содержат функциональные блоки, тогда они будут оцениваться в порядке сверху вниз, и их выходы будут объединяться в соответствии с логической операцией OR (ИЛИ). Если нет ветвей, содержащих функциональный блок, тогда будет выполняться обычная операция OR.

Для вставки параллельного ветвления с SCE функцией, выберите функциональный блок и выполните команду **Insert Contact Parallel above** (вставка выше) или **Insert Contact Parallel below** (вставка ниже). Это возможно, но только если первый вход и главный выход функционального блока имеют тип BOOL.

Ниже приведен пример генерированной модели языка для заданной цепи.

Пример SCE

Экземпляр функционального блока `x1 (TON)` имеет логический вход и логический выход. Его выполнение может быть пропущено, если условие в параллельной ветви оценивается как TRUE. Это значение условия получается из операторов `OR` и `AND` соединения контактов `cond1`, `cond2` и `cond3`.

Параллельное ветвление SCE в цепи ступенчатой диаграммы

Обработка производится, как показано в следующем примере, где `P_IN` и `P_OUT` представляют логические значения входа (точка разделения) и выхода (точка объединения) параллельного ветвления, соответственно.

```
P_IN := b1 AND b2;
IF ((P_IN AND cond1) AND (cond2 OR cond3)) THEN
   P_OUT := P_IN;
ELSE
   x1(IN := P_IN, PT := {p 10}t#2s);
   tElapsed := x1.ET;
   P_OUT := x1.Q;
END_IF
bRes := P_OUT AND b3;
```

Следующие изображения показывают поток данных (голубой) в случае, если функциональный блок выполняется (результат условий от `cond1`, `cond2` и `cond3` равен FALSE) или шунтирован (условие равно TRUE).

Условие=FALSE, функциональный блок выполняется:

Условие=TRUE, функциональный блок шунтируются:

Set/Reset в FBD/LD/IL

FBD и LD

Логический выход катушки в FBD *(см. стр. 252)* или соответственно LD *(см. стр. 253)* может быть установлен или сброшен. Для изменения между установленными состояниями используйте соответствующие команды **Set/Reset** из контекстного меню, когда выбран выход. Выход или катушка будут отмечены как `S` или `R`.

Set	Если значение TRUE достигает выхода или катушки set, этот выход/катушка становится равной TRUE и остаётся TRUE. Это значение не может быть перезаписано в этой позиции, пока приложение работает.
Reset	Если значение TRUE достигает выхода или катушки set, этот выход/катушка становится равной FALSE и остаётся FALSE. Это значение не может быть перезаписано в этой позиции, пока приложение работает.

Выход Set в FBD

В LD редакторе Вы можете вставлять катушки set и reset методом drag и drop. Для осуществления этой акции используйте или **ToolBox**, категория **Ladder elements**, или элементы `S` и `R` из панели инструментов.

Пример:

Катушка Set, катушка reset

Дальнейшую информацию см. Катушки Set/Reset *(см. стр. 293)*.

IL

В Instruction List используйте операторы S и R *(см. стр. 256)* для установки или сброса операнда.

Катушки Set/Reset

Обзор

Катушки *(см. стр. 296)* могут также быть определены как катушки set или reset.

Вы можете распознать катушку set (установки) по символу `S: (S)`. Set катушка не может перезаписать значение TRUE в соответствующей логической переменной. Это значит, что переменная, однажды установленная в TRUE остаётся TRUE.

Вы можете распознать катушку reset (сброса) по символу `R: (R)`. Reset катушка не может перезаписать значение FALSE в соответствующей логической переменной. Это значит, что переменная, однажды установленная в FALSE остаётся FALSE.

В LD редакторе Вы можете вставлять катушки set и reset непосредственно методом drag и drop из **ToolBox**, категория **Ladder elements**.

Катушка Set, катушка reset

Раздел 10.3

Элементы LD

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Контакт	295
Катушка	296

Контакт

Обзор

Это элемент LD.

В LD *(см. стр. 253)* в её левой части каждая цепь содержит 1 или несколько контактов. Контакты представляются двумя вертикальными параллельными линиями.

Контакты передают условие ON (TRUE) или OFF (FALSE) слева направо. Логическая переменная назначается для каждого контакта. Если эта переменная равна TRUE, условие передаётся слева направо и финально к катушке в правой части цепи. Иначе правое соединение получает значение FALSE.

Вы можете соединять множество контактов последовательно, так же как и параллельно. Контакты, соединённые в параллель, представляют логическое условие 'OR' (ИЛИ), так что только один из них нужен в состоянии TRUE, чтобы передать через параллельное соединение значение TRUE. С другой стороны, соединённые последовательно контакты представляют условие логическое 'AND' (И), так что все контакты должны быть TRUE, чтобы получить передачу финальным контактом значения TRUE.

Следовательно, расположение контактов соответствует электрическим или параллельным, или последовательным цепям.

Контакт также может быть инверсным. Это индицируется наклонной чертой в символе контакта.

Инверсный контакт пропускает приходящее условие (TRUE или FALSE), только если назначенная логическая переменная равна FALSE.

Вы можете вставлять контакт в LD цепь через одну из команд **Insert Contact** или **Insert Contact (right-** справа**) Insert Contact Parallel (above - выше)**, **Insert Contact Parallel (below** - **ниже)**, **Insert Rising Edge Contact** (нарастающий фронт), или **Insert Falling Edge Contact** (спадающий фронт), которые являются частью **LD** меню. Альтернативно Вы можете вставить элемент через перетаскивание из **ToolBox** *(см. стр. 280)* или из другой позиции внутри редактора (drag and drop).

FBD и IL

Если Вы работаете в виде FBD *(см. стр. 252)* или IL *(см. стр. 254)* команда не будет доступна. Но контакты и катушки, вставленные в LD цепь, будут представлены соответствующими элементами или IL инструкциями.

Катушка

Обзор

Это элемент LD.

В правой части LD цепи может быть любое число катушек, которые представлены круглыми скобками.

Они могут быть расположены только в параллель. Катушка передаёт значение соединений слева направо и копирует его в соответствующую логическую переменную. В линии записи могут присутствовать значение ON (TRUE) или значение OFF (FALSE).

Катушка также может быть инвертирована. Это индицируется наклонной чертой в символе катушки.

В этом случае инверсное значение входящего сигнала будет скопировано в соответствующую логическую переменную.

Вы можете вставить катушку в цепь через одну из команд **Insert Coil**, **Insert Set Coil**, **Insert Reset Coil**, или **Insert Negated Coil** в **LD** меню. Альтернативно Вы можете вставить элемент через перетаскивание из **ToolBox** (**Ladder elements**) или перетаскиванием из другой позиции внутри редактора. Также обращайтесь к *Катушкам Set и Reset (см. стр. 293)*.

FBD и IL

Если Вы работаете в виде FBD *(см. стр. 252)* или IL *(см. стр. 254)* команда не будет доступна. Но контакты и катушки, вставленные в LD цепь, будут представлены соответствующими элементами или IL инструкциями.

Глава 11

Редактор Continuous Function Chart (CFC)

Содержание этой главы

Эта глава содержит следующие темы:

Тема	Страница
Язык Continuous Function Chart (CFC)	298
Редактор CFC	299
Положения курсора в CFC	301
CFC элементы / Панель инструментов	303
Работа в CFC редакторе	309
CFC редактор в онлайн режиме	312
Ориентированный на страницы CFC редактор	314

Язык Continuous Function Chart (CFC)

Обзор

Continuous Function Chart (Непрерывная функциональная диаграмма) является расширением стандарта IEC 61131-3, и является графическим языком программирования, основанным на языке Function Block Diagram (FBD) *(см. стр. 252)*. Однако, в отличие от языка FBD в CFC нет цепей. CFC позволяет свободное расположение графических элементов, которое, в свою очередь, допускает обратные связи.

О создании объектов программирования в SoMachine смотрите описание CFC редактора *(см. стр. 299)*.

Редактор CFC

Обзор

Редактор CFC – это графический редактор, доступный для программирования объектов в языке программирования continuous function chart (CFC) *(см. стр. 298)*, который является расширением языков программирования IEC 61131-3. Выберите язык при добавлении Вами нового объекта program organization unit (POU) в Ваш проект. Для больших проектов рассмотрите использование ориентированной на страницы версии *(см. стр. 314)*.

Редактор будет доступен в нижней части окна, которое открывается при открытии объекта CFC POU. Это окно в своей верхней части включает также редактор деклараций (объявлений) *(см. стр. 372)*.

Редактор CFC

Реактор CFC в отличие от редактора FBD / LD позволяет свободное позиционирование *(см. стр. 309)* элементов, что позволяет прямую вставку в части обратной связи. Последовательность обработки определяется списком, который содержит все вставленные в настоящее время элементы и может быть модифицирован.

Следующие элементы доступны в панели инструментов *(см. стр. 303)* и могут быть вставлены способом перетаскивания:

блоки (операторы, функции, функциональные блоки и программы)

- вход
- выход
- переход
- метка
- возврат
- компоновщик
- селектор
- метки соединения
- комментарии

Вы можете соединять выводы входов и выходов элементов рисованием линии с помощью мыши. Путь для соединительной линии будет создан автоматически и будет следовать самому короткому маршруту. Линии соединения будут автоматически подстраиваться при перемещении элементов. За дальнейшей информацией обращайтесь к описанию вставки и размещению элементов *(см. стр. 309)*. Для сложных диаграмм Вы можете использовать соединительные метки *(см. стр. 304)* вместо линий. Вы можете также рассмотреть возможность изменения маршрута.

Может случиться, что элементы располагаются на пути так, что они перекрывают уже размещённых соединений. Эти коллизии указываются красными соединительными линиями. Если есть какие либо коллизии в диаграмме, кнопка в правом верхнем углу вида редактора получает красный контур: . Для редактирования коллизий шаг за шагом нажмите эту кнопку и выполните команду **Show next collision**. Тогда будет выделена следующая найденная затронутая связь.

Для сложных диаграмм Вы можете использовать соединительные метки *(см. стр. 304)* вместо линий. Вы можете также захотеть использовать ориентированную на страницы версию редактора CFC.

Функция масштабирования (zoom) позволяет Вам изменять размерность окна редактора: используйте кнопку в нижнем правом углу окна и выберите значение из перечисленных масштабов. Альтернативно Вы можете выбрать поле **...** , чтобы открыть диалоговое окно, где Вы можете ввести любой желаемый масштаб.

Вы можете вызывать команды работы с CFC редакторе из контекстного меню или из меню **CFC**, которое доступно, как только CFC редактор активен.

Позиции курсора в CFC

Обзор

Позиции курсора в CFC программы индицируются серым фоном, когда он находится над программным элементом.

Когда Вы нажмёте одну из этих затенённых областей, до отпускания кнопки мыши цвет фона будет изменён на красный. До тех пор пока Вы не отпустили кнопку мыши, она станет текущей позицией курсора, с выделением и отображением красным цветом соответствующего элемента или текста.

Есть 3 категории позиции курсора. Рассмотрите возможные позиции, индицируемые областями серого цвета, как показано на иллюстрациях в следующих параграфах.

Позиции курсора в тексте

Если курсор позиционируется на тексте и вы нажмёте кнопку мыши, он отображается как выделенный голубой тенью и может редактироваться. Кнопка **...** доступна для открытия помощника ввода. Вначале после вставки элемента символы ??? представляют имя элемента. Замените эти символы допустимым идентификатором. После этого подсказка отображается при позиционировании курсора на имени переменной или параметре блока. Подсказка содержит тип переменной или параметра и, если существует, связанный комментарий во второй линии.

Возможные позиции курсора и примеры выделенного текста:

Курсор, позиционированный на теле элемента

Если курсор позиционируется на теле элемента (блок, вход, выход, переход, метка, возврат, комментарий, маркер соединения), это будет отображено красным цветом и может перемещаться перемещением мыши.

Возможные позиции курсора и пример выделения тела:

Курсор, позиционированный на теле входе или выходе соединения элемента

Если курсор позиционирован на входе или выходе соединения элемента, красный заполненный квадрат будет указывать эту позицию (точку соединения). Она может быть инвертирована или установлена/сброшена (set/reset).

Возможные позиции курсора (затенённые серым) и примеры выбранных позиций выходов и входов (красные квадраты):

CFC элементы / Панель инструментов

Обзор

Графические элементы, доступные для программирования в окне CFC редактора *(см. стр. 299)* обеспечиваются панелью инструментов. Откройте панель инструментов в окне просмотра выполнением команды **ToolBox** в меню **View**.

Выберите желаемый элемент в панели инструментов и вставьте *(см. стр. 309)* его в окно редактора перетаскиванием (drag and drop).

Кроме элементов программирования есть запись

вверху списка панели инструментов. До тех пор, пока выбрана эта запись, курсор имеет вид стрелки, и Вы можете выбирать элементы в окне редактора для позиционирования и редактирования.

Элементы CFC

Имя	Символ	Описание
page (страница)	1 Overview	Номер страницы задаётся автоматически согласно её расположению. Вы можете вводить имя (`Overview` в этом примере) в оранжевом поле вверху станицы.
control point (контрольная точка)		Контрольная точка необходима для фиксации измененного вручную направления линии связи. Это поможет предотвратить изменения, производимые командой **Route all Connections**. Двумя контрольными точками Вы можете отметить определённый сегмент линии, для которого вы хотите изменить маршрут.
input	???	Вы можете выбрать текст, представленный как `???` и заменить его переменной или константой. Помощник ввода обеспечивает выбор допустимого идентификатора.
output	???	
box (блок)	???	Вы можете использовать блок для представления операторов, функций, функциональных блоков и программ. Вы можете выбрать текст, представленный как `???` и заменить его именем оператора, функции, функционального блока и программы. Помощник ввода обеспечивает выбор одного из доступных объектов. Если Вы выбрали функциональный блок, другое поле `???` будет показано выше блока. Замените знаки вопроса именем экземпляра функционального блока. Если Вы заменяете существующий блок на другой (изменением введённого имени типа блока), и новый имеет отличное минимальное или максимальное число выводов входов или выходов, выводы будут соответственно адаптированы. Если выводы удаляются, нижний будет удалён первым.
jump (переход)	???	Используйте элемент перехода, чтобы указать в какой позиции должно продолжиться исполнение программы. Эта позиция определяется меткой (см. ниже). Следовательно, замените текст, представленный как `???` именем метки.
label (метка)	??? .	Метка помечает позицию, в которую программа может перейти (см. элемент jump). В онлайн режиме метка возврата для отметки конца POU вставляется автоматически.
return (возврат)	RETURN	В онлайн режиме элемент возврата автоматически вставляется в первой колонке и после последнего элемента в редакторе. При пошаговом выполнении он автоматически переходит к выполняемому прежде, покидая POU.
composer (компоновщик)	???	Используйте компоновщик для поддержки входа блока, тип которого является структурой. Компоновщик будет отображать компоненты структуры и, следовательно, делать их доступными в CFC для программиста. Для этой цели назовите компоновщика связанной структурой (заменой `???` на имя) и соедините его с блоком, вместо использования элемента входа.

Имя	Символ	Описание
selector (селектор)		Селектор в противоположность компоновщику используется для поддержки выхода блока, тип которого – структура. Селектор будет отображать компоненты структуры и, следовательно, делать их доступными в CFC для программиста. Для этой цели назовите компоновщика связанной структурой (заменой `???` на имя) и соедините его с блоком, вместо использования элемента выхода.
comment (комментарий)		Используйте этот элемент для добавления любых комментариев в диаграмму. Выберите шаблон текста и замените его желаемым текстом. Для получения новой линии внутри комментария нажмите CTRL + ENTER.
connection mark – source (источник) connection mark – sink (приёмник)	 	Вы можете использовать знак соединения вместо линии соединения *(см. стр. 310)* между элементами. Это может помочь прояснить сложные диаграммы. Для допустимого соединения назначьте знак соединения – элемент источника на выход элемента и назначьте знак соединения – приёмник (см. ниже) на вход другого элемента. Назначьте одинаковое имя для обоих знаков (без разделения заглавных и строчных символов). Наименование: Первый элемент connection mark – source, вставленный в CFC, по умолчанию именуется `C-1` и может быть изменён вручную. В его дубликате connection mark – sink замените `???` той же строкой имени, что и в source mark (источнике). Редактор проверит, что имена меток уникальны. Если имя метки источника изменяется, имя связанной метки приёмника будет также автоматически изменяться. Однако, если sink mark переименовывается, source mark сохранит старое имя. Это позволяет Вам переконфигурировать соединения. Аналогично, удаление метки соединения не удаляет её дубликата. Чтобы использовать метку соединения, перетащите её из панели инструментов в окно редактора и затем соедините её вывод с выходом или входом соответствующего элемента. Альтернативно Вы можете конвертировать в существующую линию нормального соединения, используя команду **Connection Mark**. Эта команда позволяет Вам изменить метки соединения назад к нормальным линиям соединения. За иллюстрациями, показывающими некоторые примеры меток соединения, обращайтесь к главе ***Метки соединения.***
input pin (вывод входа)		В зависимости от типа блока Вы можете добавлять дополнительные входы. Для этой цели выберите элемент блока в цепи CFC и нарисуйте элемент входного вывода в блоке.
output pin (вывод выхода)	–	В зависимости от типа блока Вы можете добавлять дополнительные входы. Для этой цели выберите элемент блока в цепи CFC и нарисуйте элемент выходного вывода в блоке.

Пример компоновщика

CFC программа `cfc_prog` поддерживает экземпляр функционального блока `fublo1`, который имеет входную переменную `struvar` типа structure. Используйте элемент компоновщика (composer) для доступа к компонентами структуры.

Определение структуры `stru1` :

```
TYPE stru1 :
STRUCT
  ivar:INT;
  strvar:STRING:='hallo';
END_STRUCT
END_TYPE
```

Объявление и реализация функционального блока `fublo1`:

```
FUNCTION_BLOCK fublo1
VAR_INPUT
  struvar:STRU1;
END_VAR
VAR_OUTPUT
  fbout_i:INT;
  fbout_str:STRING;
END_VAR
VAR
  fbvar:STRING:='world';
END_VAR
fbout_i:=struvar.ivar+2;
fbout_str:=CONCAT (struvar.strvar,fbvar);
```

Объявление и реализация программы `cfc_prog`: `PROGRAM cfc_prog`

```
VAR
  intvar: INT;
  stringvar: STRING;
  fbinst: fublo1;
  erg1: INT;
  erg2: STRING;
END_VAR
```

Элемент компоновщика (composer)

1 composer
2 функциональный блок с входной переменной `struvar` типа структуры `stru1`

Пример селектора

CFC программа `cfc_prog` поддерживает экземпляр функционального блока `fublo2`, который имеет выходную переменную `fbout` типа structure `stru1`. Используйте элемент селектора (selector) для доступа к компонентами структуры.

Определение структуры `stru1`:

```
TYPE stru1 :
STRUCT
  ivar:INT;
  strvar:STRING:='hallo';
END_STRUCT
END_TYPE
```

Объявление и реализация функционального блока `fublo1`:

```
FUNCTION_BLOCK fublo2
VAR_INPUT CONSTANT
  fbin1:INT;
  fbin2:DWORD:=24354333;
  fbin3:STRING:='hallo';
END_VAR
VAR_INPUT
  fbin : INT;
END_VAR
VAR_OUTPUT
  fbout : stru1;
  fbout2:DWORD;
END_VAR
VAR
  fbvar:INT;
  fbvar2:STRING;
END_VAR
```

Объявление и реализация программы `cfc_prog`:

```
VAR
  intvar: INT;
  stringvar: STRING;
  fbinst: fublo1;
  erg1: INT;
  erg2: STRING;
  fbinst2: fublo2;
END_VAR
```

Иллюстрация показывает элемент селектора, где неиспользуемые выводы были удалены с помощью выполнения команды **Remove Unused Pins**.

1 функциональный блок с выходной переменной `fbout` типа structure `stru1`
2 selector

Работа в CFC редакторе

Обзор

Элементы, доступные для программирования в CFC редакторе, обеспечены в панели инструментов **ToolBox** *(см. стр. 303)*, которая по умолчанию доступна в окне, как только CFC редактор открыт. **Tools** → **Options** → **CFC editor** определяет основные уставки для работы с редактором.

Вставка

Для вставки элемента выберите его в **ToolBox** *(см. стр. 303)* с помощью нажатия кнопки мыши, держа кнопку нажатой, перетащите элемент в желаемую позицию в окне редактора. Во время перетаскивания курсор будет отображаться как стрелка плюс прямоугольник со знаком плюса. При реализации (отпускании кнопки) элемент будет вставлен.

Выбор

Чтобы выбрать вставленный элемент для дальнейших действий, таких как редактирование или перерасположения, нажмите на тело элемента для его выделения. Он будет отображён по умолчанию как затенённым красным. Дополнительным нажатием клавиши SHIFT Вы можете нажимать и выделять последующие элементы. Вы можете также нажать левую кнопку мыши и нарисовать пунктирный прямоугольник вокруг всех элементов, которые Вы хотите выделить. Как только Вы отпустите кнопку, будет показано выделение. Командой **Select all** сразу выделяются все элементы.

Использованием клавиш стрелок Вы можете сдвигать метку выделения к следующей возможной позиции курсора. Последовательность зависит от порядка выполнения или элементов, который индицируется номерами элементов *(см. стр. 311)*.

Когда выбран вывод входа и Вы нажимаете CTRL + LEFT ARROW (стрелку влево), будет выделен соответствующий выход. Когда выделен вывод выхода и Вы нажимаете CTRL + LEFT ARROW, будут выделены соответствующие выходы.

Замена блоков

Чтобы заменить существующий элемент блока, замените текущий вставленный идентификатор на тот, который имеет желаемый новый элемент. Число входов и выходов будет адаптировано, если это необходимо из-за определения POUs и, следовательно, некоторые существующие назначения должны быть удалены.

Перемещение

Чтобы переместить элемент, выберите элемент нажатием на тело элемента (см. возможные позиции курсора *(см. стр. 301)*) и перетащите его при нажатой кнопке мыши в желаемую позицию. Затем отпустите кнопку мыши для размещения элемента. В также можете использовать команды Cut и Paste (вырезание и вставка) для этой цели.

Соединение

Вы можете соединять выводы входа и выхода двух элементов соединительной линией или через метки соединения.

Соединительная линия: Вы можете или выбрать допустимую точку соединения, которая является выводом входа или выхода элемента (обращайтесь к *Позициям курсора в CFC* *(см. стр. 301)*), и затем провести линию к другой точке соединения с помощью мыши. Или Вы можете выбрать 2 точки соединения и выполнить команду **Select connected pins**. Выбранная возможная точка соединения указывается красным заполненным квадратом. Когда Вы проводите линию от такой точки к целевому элементу, Вы можете идентифицировать возможную целевую точку соединения.

На следующем рисунке приведен пример: После нажатия мыши на выводе входа `var1` элемента красный прямоугольник отображается, показывая, что это выбранная точка соединения. Удерживая кнопку нажатой, переместите курсор к выводу выхода блока ADD, пока символ курсора не станет таким, как показано на рисунке. Теперь отпустите кнопку мыши для установления соединения.

Кратчайшее возможное соединение будет создано, принимая во внимание другие элементы и соединения. Если маршрут линий соединения пересекает другие линии соединения, они выделяются светло-серым цветом.

Метки соединения: Вы могли бы также использовать метки соединения вместо соединительных линий с целью упрощения сложных диаграмм. Обращайтесь к описанию меток соединения *(см. стр. 304)*.

Копирование

Для копирования элемента выберите его и используйте команды **Copy** и **Paste**.

Редактирование

После того как Вы вставили элемент, по умолчанию текстовая часть представляется как `???`. Чтобы заменить это желаемым текстом (именем POU, именем метки, именем экземпляра, комментарием и т.п.), нажмите текст для редактирования поля. Также кнопка **…** будет доступна для открытия **Input Assistant**.

Удаление

Вы можете удалить выделенный элемент или линию соединения выполнением команды **Delete**, которая доступна в контекстном меню, или нажатием клавиши DEL.

Открытие функционального блока

Если функциональный блок добавлен в редактор, Вы можете открыть этот блок двойным нажатием. Альтернативно используйте команду **Browse** → **Go To Definition** из контекстного меню.

Порядок выполнения, номера элементов

Последовательность, в которой элементы цепи CFC выполняются в онлайн режиме, индицируется номерами в верхнем правом углу блока, выхода, перехода, возврата и элементов меток. Выполнение начинается с элемента с наименьшим номером, который равен 0.

Вы можете изменять порядок выполнения команд, которые возможны в подменю **Execution Order** меню **CFC**.

Когда элемент добавляется, число будет автоматически задаваться согласно топологической последовательности (слева направо и сверху вниз). Новый элемент получает номер его топологического приемника, если последовательность уже была изменена, и все старшие номера увеличиваются на 1.

Обратите внимание на то, что номер элемента остаётся постоянным при его перемещении.

Обратите внимание на то, что последовательность влияет на результат и в определённых случаях должна быть изменена.

Изменение размера рабочей страницы

Для того, чтобы получить больше пространства вокруг существующей CFC диаграммы в окне редактора, вы можете изменить размер рабочей области (рабочей страницы). Произведите это с помощью выбора и перетаскивания всех элементов мышью или команд вырезания и вставки (обращайтесь к *Перемещению* (*см. стр. 309*)).

Альтернативно Вы можете использовать специальное диалоговое окно уставок размера. Это может сократить время в случае больших диаграмм. Обращайтесь к описанию диалогового окна **Edit Working Sheet** *(see SoMachine, Menu Commands, Online Help)*. В случае ориентированной на страницы CFC Вы можете использовать команду **Edit Page Size** *(see SoMachine, Menu Commands, Online Help)*.

Редактор CFC в онлайн режиме

Обзор

В онлайн режиме CFC редактор обеспечивает виды для мониторинга. Виды для записи и форсирования переменных и выражений в контроллере описаны в отдельных главах. Функциональность отладки (точки остановки, пошаговое выполнение и т.п.) доступны, как описано ниже.

- За информацией о том, как открыть объекты в онлайн режиме, обращайтесь к описанию интерфейса пользователя в онлайн режиме *(см. стр. 49)*.
- Окно редактора CFC объекта также содержит редактор деклараций (объявлений) в верхней части. Обращайтесь к описанию редактора деклараций в онлайн режиме *(см. стр. 376)*.

Мониторинг

Актуальные значения отображаются в маленьких окнах мониторинга за каждой переменной (мониторинг в линии).

Онлайн вид объекта программы PLC_PRG:

Положение точек останова в CFC редакторе

Возможные позиции точек останова - обычно те позиции POU, в которых значения переменных могут изменяться, или в которых программный поток разветвляется, или вызывается другая POU. Смотрите возможные позиции на следующем рисунке.

Позиции точек остановок (breakpoint) в CFC редакторе:

ПРИМЕЧАНИЕ: Точка остановки будет установлена автоматически во всех методах, которые могут быть вызваны. Если вызывается метод, управляемый интерфейсом, точки останова будут установлены во всех методах функциональных блоков, реализующих этот интерфейс, и во всех производных блоках, подписанных на метод. Если метод вызывается через указатель на функциональный блок, точки останова будут установлены в методе функционального блока, и во всех производных функциональных блоках, подписанных на метод.

Ориентированный на страницы CFC редактор

Обзор

В дополнение к стандартному CFC редактору SoMachine обеспечивает CFC редактор с разбиением на страницы. Помимо инструментов *(см. стр. 303)* и команд стандартного CFC редактора этот редактор позволяет Вам располагать элементы на любом числе разных страниц.

ПРИМЕЧАНИЕ: Вы не можете конвертировать POUs, созданные в CFC с разбивкой на страницы в нормальную CFC и наоборот. Вы можете копировать элементы между этими двумя редакторами командами copy и paste (через клипборд - буфер обмена) или функцию drag и drop.

CFC с разбивкой на страницы

Для изменения размера страницы выполните команду **Edit Page Size**.

Соединения между двумя страницами

Соединение между 2 страницами реализуется с элементами меток соединения - source и connection mark – sink (обращайтесь к описанию меток соединения). Вы можете поместить метку соединения - source претаскиванием к правой границе - метку соединения – sink к левой границе. Если Вы проводите соединительную линию от входа или выхода элемента до границы, метка соединения помещается автоматически.

Вставка соединительных меток

Порядок выполнения

Порядок выполнения страниц сверху вниз. Внутри страницы порядок следует правилам стандартного CFC редактора (обращайтесь к дополнительной информации по порядку выполнения *(см. стр. 311)*). Вы можете изменять порядок выполнения элементов только внутри связанной страницы. Вы не можете изменять порядок выполнения элементов на других страницах.

Глава 12

Редактор Sequential Function Chart (SFC)

Содержание этой главы

Эта глава содержит следующие темы:

Тема	Страница
SFC редактор	318
Язык SFC - Sequential Function Chart	320
Позиции курсора в SFC	321
Работа в SFC редакторе	323
Свойства элемента SFC	325
SFC элементы / Панель инструментов	327
Определитель для действий в SFC	338
Неявные переменные - SFC флаги	339
Последовательность выполнения в SFC	344
SFC редактор в онлайн режиме	346

SFC редактор

Обзор

Редактор SFC доступен для программирования объектов в стандарте IEC 61131-3 языка программирования SFC - Sequential Function Chart (Последовательных функциональных диаграмм) *(см. стр. 320)*. Выбирайте язык, когда Вы добавляете новый объект в проект.

Редактор SFC – графический редактор. Произведите основные уставки, касающиеся поведения и отображения в диалоговом окне **Options** → **SFC editor**.

Редактор SFC доступен в нижней части окна, которое открывается, когда Вы редактируете объект SFC POU. Это окно также содержит **Declaration Editor** *(см. стр. 372)* в верхней части.

Редактор SFC

Работа с SFC редактором

Элементы *(см. стр. 327)*, используемые в SFC диаграмме, доступны в **SFC** меню. Меню доступно, как только SFC редактор активен. Располагайте их в последовательных или параллельных последовательностях шагов, которые соединены через переходы. За дальнейшей информацией обращайтесь к *Работе в SFC редакторе (см. стр. 323)*.

Вы можете редактировать свойства шагов в отдельном окне свойств *(см. стр. 325)*. Среди другого Вы можете определять минимальное и максимальное время активности каждого шага.

Вы можете получить доступ к неявным переменным *(см. стр. 339)* для контролирования выполнения SFC (например, статус шага, анализирование таймаута, перезапуск).

Язык SFC - Sequential Function Chart

Обзор

Sequential Function Chart (SFC) – графически ориентированный язык, который описывает хронологический порядок конкретных действий внутри программы. Эти действия возможны как отдельные программные объекты, написанные на любом доступном языке программирования. В SFC они назначены на элементы шагов, и последовательность выполнения управляется элементами переходов. За детальным описанием того, как шаги будут выполняться в онлайн режиме, обратитесь к *Последовательности выполнения в SFC* (*см. стр. 344*).

За информацией о том, как использовать SFC редактор в SoMachine, обращайтесь к *SFC редактору* (*см. стр. 318*).

Пример

Пример последовательности шагов в SFC модуле:

Позиции курсора в SFC

Обзор

Возможные позиции курсора в SFC диаграмме в SFC редакторе *(см. стр. 318)* индицируются серой тенью, при перемещении курсора над элементами.

Позиции курсора в текстах

Есть две категории позиций курсора: тексты и элементы тела. Смотрите возможные позиции, указанные серыми затенёнными областями, как показано на следующих иллюстрациях:

Возможные позиции курсора в текстах:

При нажатии текстовой позиции курсора строка становится редактируемой.

Выбор имени действия для редактирования:

Позиции курсора в элементах тела

Возможные позиции курсора в элементах тела:

Когда Вы нажимаете затенённую область, элемент выбирается. Он получает пунктирный контур и отображается затенённым красным цветом (для многократного выделения обращайтесь к *Работе с SFC редактором (см. стр. 323)*).

Выбранный элемент шага

Работа в SFC редакторе

Обзор

По умолчанию новая SFC POU содержит шаг инициализации и последовательный переход. Эта глава даёт информацию о том, как добавлять последующие элементы и как располагать и редактировать элементы.

Возможные позиции курсора

За дальнейшей информацией обращайтесь к разделу *Позиции курсора в SFC (см. стр.321)*.

Навигация

Используйте клавиши стрелок для переходов к следующему или предыдущему элементу диаграммы.

Вставка элементов

Для вставки конкретных SFC элементов *(см. стр. 327)*, выполните соответствующие команды из **SFC** меню. За дальнейшей информацией обращайтесь к описанию команд SFC редактора *(см. SoMachine, Menu Commands, Online Help)*. Дважды нажмите уже вставленный шаг, переход или элемент действия (action), который ещё не связан с объектом программирования, чтобы открыть диалоговое окно для назначения.

Выбор элементов

Выберите элемент или текстовое поле нажатием возможной позиции курсора. Вы также можете получить выделение смежного элемента использованием клавиш стрелок. Элемент будет изменять цвет на красный. Например, смотрите раздел *Позиции курсора в SFC (см. стр.321)*.

ПРИМЕЧАНИЕ: В отличие от предыдущей версии SoMachine, Вы можете выбирать и, следовательно, также перемещать (вырезать, копировать, вставлять) или удалять шаги и переходы по отдельности.

Для многократного выделения доступны следующие возможности:
- Держите клавишу SHIFT нажатой и затем нажимайте на конкретные элементы, которые нужно выбрать.
- Нажмите левую кнопку и создайте прямоугольник вокруг элементов, которые нужно выбрать.
- Выполните команду **Select All**, по умолчанию из **Edit** меню.

Редактирование текстов

Нажмите текстовую позицию курсора, чтобы открыть поле редактирования, где Вы можете редактировать текст. Если текстовая область была выбрана с помощью клавиш стрелок, откройте поле редактирования явно, используя клавишу пробела SPACE.

Редактирование связанных действий

Дважды нажмите ассоциацию действия шага (входа, активного или выхода) или перехода, чтобы открыть ассоциированное действие в соответствующем редакторе. Вы можете, например, дважды нажать элемент перехода или треугольник, указывающий на действие при выходе в элементе шага.

Вырезание, копирование, вставка элементов

Выберите элементы и выполните команду **Cut**, **Copy** или **Paste** (из **Edit** меню) или используйте соответствующие клавиши.

ПРИМЕЧАНИЕ:

- Когда Вы вставляете один или несколько вырезанных или скопированных элементов, содержимое буфера обмена будет вставлено перед выбранной в данный момент позицией. Если ничего не выбрано, элементы будут добавлены в конце загруженной в данный момент диаграммы.
- Если Вы вставляете ветвление, когда выбранный текущий элемент также является ветвлением, вставляемые элементы ветвления будут вставлены слева от существующих элементов.
- Если Вы вставляете действие (список) в текущем выбранном шаге, действия будут добавлены в начале списка действий шага или будет создан список действий шага.
- Несовместимые элементы при вырезании/вставке:
  Если Вы выделяете ассоциированное действие (список) и дополнительно элемент, который не является шагом, к которому действие (список) принадлежат, будет выводиться сообщение: **The current selection contains incompatible elements. No data will be filed to the clipboard.** Выделение не будет сохранено, и Вы не сможете вставить или копировать его куда-либо ещё.
- Несовместимые элементы при вставке:
  Если Вы пытаетесь вставить действие (список), когда текущий выбранный элемент не является шагом или другая ассоциация, будет отображаться сообщение: **The current clipboard content cannot be pasted at the current selection.** Если Вы попробуете вставить элемент, такой как шаг, ветвление или переход, когда в данный момент выделено действие (список), будет отображаться такое же сообщение.

Удаление элементов

Выберите элементы и выполните команду **Delete** или нажмите клавишу DEL.
Рассмотрите следующее:

- Удаление шага также удаляет ассоциированный список действий.
- Удаление начального шага автоматически устанавливает следующий шаг как начальный. Эта опция **Initial step** будет активирована в свойствах этого шага.
- Удаление горизонтальной линии, предшествующей области ветвления, удалит все ветвления.
- Удаление всех конкретных элементов ветвления будет удалять ветвление.

Свойства SFC элемента

Обзор

Вы можете видеть и редактировать свойства SFC элемента в диалоговом окне **Element Properties**. Откройте это диалоговое окно через команду **Element Properties**, которая является частью меню **View**.

То, какие свойства отображаются, зависит от выбранного в данный момент элемента. Свойства сгруппированы. Вы можете открывать или закрывать конкретную группу секций, используя знаки плюса или минуса.

Вы можете сконфигурировать должны ли конкретные типы свойств отображаться следом за элементом в SFC диаграмме из вкладки **View** опций SFC редактора.

Общие свойства

Свойство	Описание
Name (Имя)	Имя элемента, по умолчанию <элемент><номер выполнения> Примеры: имя шага Step0, Step1, имя ветвления branch0 и т.п.
Comment (Комментарий)	Комментарий к элементу, текстовая строка Пример: Reset the counter. Press CTRL + ENTER to insert line breaks.
Symbol (Символ)	Для каждого SFC элемента неявно создаётся флаг, именуемый как элемент. Здесь вы можете специфицировать должна ли эта переменная флага экспортироваться в конфигурацию символов и как символ затем должен быть доступным в контроллере. Дважды нажмите поле значения или выберите поле значения и нажмите клавишу пробела с целью открытия списка выбора, из которого Вы сможете выбрать одну из следующих опций доступа: None: Символ будет экспортирован в конфигурацию символов, но не будет доступен в контроллере. Read: Символ будет экспортирован в конфигурацию символов, и он будет читабелен для контроллера. Write: Символ будет экспортирован в конфигурацию символов, и он будет записан в контроллере. Read/Write: Комбинация чтения и записи. По умолчанию это поле пустое слева. То есть, нет символов, экспортируемых в конфигурацию.

Специальные свойства

Специальные свойства		Описание
Initial step (начальный шаг)		Эта опция активируется в свойствах текущего начального шага (init step) *(см. стр. 327)*. По умолчанию она активирована для первого шага в SFC и деактивирована для других шагов. Если Вы активируете эту опцию для других шагов, Вы должны деактивировать её у предыдущего начального шага. Иначе компилятором будет сгенерирована ошибка.
Times: Времена		Определяет минимальное и максимальное время выполнения для шага. **ПРИМЕЧАНИЕ:** Тайм-ауты в шагах индицируются неявной переменной *(см. стр. 339)* SFCError флаг.
	Minimal active Минимальной активности	Минимальный интервал времени выполнения этого шага. Допустимые значения: время согласно IEC синтаксису (например, `t#8s`) или переменная TIME; по умолчанию: `t#0s`.
	Maximal active Максимальной активности	Максимальный интервал времени выполнения этого шага. Допустимые значения: время согласно IEC синтаксису (например, `t#8s`) или переменная TIME; по умолчанию: `t#0s`.
Actions: Акции - действия		Определяет действия *(см. стр. 331)*, производимые, когда шаг активен. Обращайтесь к описанию Последовательность выполнения SFC *(см. стр. 344)* за деталями.
	Step entry	Это действие будет произведено после того, как шаг станет активным.
	Step active	Это действие будет произведено, когда шаг активен и возможные входные действия уже выполнены.
	Step exit	Это действие будет произведено в последующем цикле после того, как шаг был деактивирован (действие при выходе).

ПРИМЕЧАНИЕ: Используйте соответствующие неявные переменные для определения статуса действий и тайм-аутов через SFC флаги *(см. стр. 339)*.

SFC элементы / Панель инструментов

Обзор

Вы можете использовать графические элементы, используемые для программирования в окне SFC редактора, выполнением команд из **SFC** меню.

Для информации о работе в редакторе обращайтесь к описанию в разделе *Работа в SFC редакторе*

Следующие элементы доступны и описаны в этом разделе:

- step – шаг *(см. стр. 327)*
- transition - переход*(см. стр.327)*
- action – действие *(см. стр. 331)*
- branch (alternative) – ветвление (альтернативное) *(см. стр. 334)*
- branch (parallel) – ветвление (параллельное) *(см. стр. 334)*
- jump - переход*(см. стр. 336)*
- macro – макрос *(см. стр. 336)*

Шаг / Переход

Для вставки одного шага или одного перехода выполните команду **Step** или **Transition** из **ToolBox**. Для вставки одного шага или одного перехода выполните команду **Step** или **Transition** из **ToolBox**. Шаги и переходы также могут быть вставлены в комбинации, через команды **Insert step- transition** ( ) или **Insert step-transition after** ( ) из панели инструментов.

Шаг представляется блоком, первоначально содержащим автоматически сгенерированное имя шага. Он связан линиями с предыдущим и последующим переходами. Контур блока первого шага в SFC, который является начальным шагом, показан двойной линией.

Переход представляется маленьким прямоугольником. После вставки он имеет имя по умолчанию `Trans<n>`, где `n` – номер выполнения.

Пример для шага и последующего перехода:

Пример для начального шага и последующего перехода:

Вы можете редактировать имена шага и перехода в линии.

Имена шагов должны быть уникальными в пределах родительской POU. Рассматривайте это особенно в случае, когда используете действия, программируемые в SFC. Иначе ошибка будет обнаружена во время процесса построения.

Вы можете трансформировать каждый шаг в начальный выполнением команды **Init step** или активированием соответствующего свойства шага. Начальный шаг будет выполняться первым, когда вызывается IL-POU.

Каждый шаг определён свойствами шага *(см. стр. 325)*.

После того, как Вы вставили шаг, ассоциируйте действия, которые будут выполняться, когда шаг активен (выполняется); смотрите ниже последующую информацию о действиях *(см. стр. 331)*.

Рекомендации по переходам

Переход должен обеспечить условие, при котором последующий шаг должен стать активным, как только значение условия равно TRUE. Следовательно, условие перехода должно иметь значение TRUE или FALSE.

Условие перехода может быть определено следующими двумя путями:

Тип определения	Тип условия	Описание
direct (прямое)	inline (на линии)	Замените имя перехода по умолчанию на один из следующих элементов: ● boolean variable – логическая переменная ● boolean address – логический адрес ● boolean constant - логическая константа ● инструкция, имеющая логический результат (пример: (i<100) AND b). Вы не можете здесь специфицировать программы, функциональные блоки или назначения.
использование отдельного перехода или свойства объекта	multi-use многократное использование	Замените имя перехода по умолчанию на имя объекта перехода ( ) или свойства ( ) доступного в проекте. (Это позволяет многократное использование переходов; смотрите пример `condition_xy` в рисунках ниже.) Объект, как и в переходе на линии, может содержать следующие элементы: ● boolean variable – логическая переменная ● address - адрес ● constant - константа ● instruction - инструкция ● различные выражения с произвольным кодом

ПРИМЕЧАНИЕ: Если переход создает многочисленные операторы, назначьте желаемое выражение на переменную перехода.

ПРИМЕЧАНИЕ: Переходы, которые состоят из объекта перехода или свойства, индицируются маленьким треугольником в верхнем правом углу прямоугольника.

Объект перехода (Transition object – многократное использование перехода):

Примеры переходов:

1 Условия переходов, введённые прямо
2 Переход `condition_xy`, запрограммированный на ST

Многократное использование условий (переходов или свойств) индицируется треугольником:

В отличие от предыдущей версии SoMachine, вызов перехода поддерживается как вызов метода. Он будет введён согласно следующему синтаксису:

<transition name>:=<transition condition>;

Пример: `trans1:= (a=100);`
Или только

<transition condition>; Пример: `a=100;`
Смотрите также пример (`condition_xy`) на рисунке *Примеры переходов*.

Action - Действие

Действие может содержать серию инструкций, написанных на одном из допустимых языков программирования. Оно назначено на шаг и в онлайн режиме оно будет выполняться согласно заданной последовательности выполнения *(см. стр. 344)* .

Каждое действие, используемое в шагах SFC, должно быть доступно как действительная POU внутри SFC POU или проекта ( ).

Имена шагов должны быть уникальными в пределах родительской POU. Действие не может содержать шаг, имеющий такое же имя как шаг, к которому он назначен. Иначе будет обнаружена ошибка во время построения проекта.

Пример действия, написанного на ST

Соответствующие IEC и расширяющие IEC действия шагов описаны в последующих параграфах.

Соответствующая IEC Action (IEC Action - действие)

Это действие согласно стандарту IEC61131-3, которое будет выполняться согласно его спецификатору (qualifier) *(см. стр. 338)*, когда шаг становится активным и затем второй раз, когда он становится деактивированным. В случае назначения на шаг нескольких действий, список действий будет выполняться сверху вниз.

- Различные спецификаторы могут быть использованы для IEC акций шага в отличие от нормального действия шага.
- Другое отличие от нормальных действий шага состоит в том, что каждое IEC действие шага производится с флагом управления. Это обеспечивает то, что даже если действие вызывается другим шагом, действие выполняется только один раз. Это не относится к нормальным действиям шагов.

- Действие (action) IEC шага представлено блоком из двух частей, соединённым справа от шага соединительной линией. В левой части оно показывает спецификатор действия, в правой части – имя действия. Вы можете обе части редактировать в линии.
- Действия IEC шагов связывается с шагом через команду **Insert action association**. Вы можете связать одно или несколько действий с шагом. Положение нового действия зависит от текущей позиции курсора и команды. Действие должно быть доступным в проекте и вставленным с уникальным именем действия.

Список соответствующих IEC действий, связанных с шагом:

Каждый блок в первой колонке показывает спецификатор действия и во второй – имя действия.

Расширение IEC действий шага

Это действия, расширяющие IEC стандарт. Они должны быть доступны как объекты ниже SFC объекта. Выбирайте уникальные имена действий. Они определяются в свойствах шага.

Эта таблица перечисляет расширяющие IEC действия шага:

Тип действия	Выполнение	Ассоциация	Представление
действие при входе в шаг (активация шага)	Этот тип действия шага будет выполняться, как только шаг становится активным и перед действием при активном шаге.	Действие связывается с шагом через запись в поле **Step entry** свойств шага *(см. стр. 325)*.	Оно представляется символом `E` в нижнем левом углу соответствующего блока шага.
действие при активном шаге (действие шага)	Этот тип действия шага будет выполняться, когда шаг стал активным и после возможного выполнения действия при входе в шаг. Однако, в отличие от IEC действия шага (см. выше) оно не выполняется снова при деактивации шага и не может иметь назначенных спецификаторов.	Действие связывается с шагом через запись в поле **Step active** свойств шага *(см. стр. 325)*.	Оно представляется маленьким треугольником в верхнем правом углу соответствующего блока шага.
действие при выходе из шага (деактивация шага)	действие при выходе будет выполняться один раз, когда шаг становится деактивированным. Однако, это выполнение будет производиться не в том же, а в начале последующего цикла.	Действие связывается с шагом через запись в поле **Step exit** свойств шага *(см. стр. 325)*.	Оно представляется символом `X` в нижнем правом углу соответствующего блока шага.

Расширяющие IEC действия шага

Пример: Отличие между IEC соответствующими / расширяющими действиями шага

Главное отличие между действиями шага и IEC действиями со спецификатором `N` в том, что IEC действие выполняется, по крайней мере, дважды: первый раз, когда шаг активируется, и второй раз, когда шаг деактивируется. Смотрите следующий пример.

Действие `Action_AS1` связано с шагом `AS1` как действие шага (слева), или как IEC действие со спецификатором `N` (справа). Из-за того, что в обоих случаях используются два перехода, до повторного достижения начального шага будет проходить два цикла контроллера, вызывая увеличение на 1 переменной `iCounter` в `Action_AS1`. После реактивации шага `Init`, `iCounter` в левом примере будет иметь значение 1. В правом примере, однако, он будет иметь значение 2, так как IEC действие – из-за деактивации `AS1` – было выполнено дважды.

За дальнейшей информацией по спецификаторам обращайтесь к их списку *(см. стр. 338)*.

Ветвления

Последовательные функциональная диаграмма (SFC) может разделяться; то есть линия обработки может ветвиться на 2 или несколько дальнейших линий (ветвей). Параллельные ветви *(см. стр. 334)* будут выполняться параллельно (одновременно). В случае альтернативного ветвления *(см. стр. 334)* только одно будет выполняться в зависимости от предшествующего условия перехода. Каждому ветвлению внутри диаграммы предшествует горизонтальная двойная (параллельное) или простая (альтернативное) линия, и также прерываются такими линиями или переходом *(см. стр. 336)*.

Параллельное ветвление

Параллельное ветвление должно начинаться и заканчиваться шагом. Параллельные ветвления могут содержать альтернативные ветвления и другие параллельные ветвления.

Горизонтальные линии до и после зоны ветвления – двойные линии.

Выполнение в онлайн режиме: Если предшествующий переход (`t2` в примере, показано слева) равен TRUE, первые шаги параллельного ветвления станут активными (`Step11` и `Step21`). Конкретные ветви будут выполняться параллельно друг другу перед тем как будет распознан последующий переход.

Для вставки параллельной ветви выберите шаг и выполните команду **Insert branch right**.

Вы можете преобразовать параллельные и альтернативные ветвления друг в друга выполнением команд **Parallel** или **Alternative**.

Автоматически добавляется метка ветвления в горизонтальной линии, предшествующей ветвлению, которая получает имя `Branch<n>`, где `n` - номер выполнения, начиная с 0. Вы можете специфицировать эту метку, когда определяете цель перехода *(см. стр. 336)*.

Альтернативное ветвление

Горизонтальные линии до и после зоны ветвления – простые линии.

Альтернативное ветвление должно начинаться и заканчиваться переходом. Альтернативные ветвления могут содержать параллельные ветвления и другие альтернативные ветвления.

Если шаг, который предшествует линии начала альтернативы, активен, тогда первый переход каждой альтернативной ветви будет анализироваться слева направо. Первый переход слева, чьё условие перехода равно TRUE, будет открыт, и последующие шаги будут активированы.

Для вставки альтернативного ветвления выберите переход и выполните команду **Insert branch right**.

Вы можете трансформировать параллельные и альтернативные ветвления из одного в другое выполнением команд **Parallel** или **Alternative**.

Переход к

Переход к (jump) представляется вертикальной соединительной линией плюс горизонтальная стрелка и имя цели перехода. Оно определяет следующий шаг, выполняемый, как только предшествующий переход равен TRUE. Вы можете использовать переходы, чтобы предотвратить пересечения или проведение вверх линий выполнения.

Кроме перехода по умолчанию в конце диаграммы переход может быть использован только в конце ветвления. Для вставки перехода выберите последний переход (transition) ветвления и выполните команду **Insert jump**.

Цель перехода специфицируется связанной текстовой строкой, которая может редактироваться. Это может быть имя шага или метки параллельного ветвления.

Macro -макрос

Главный вид SFC редактора

Вид редактора Macro для `Macro1`

Макрос представляется обрамлённым жирной линией блоком, содержащим имя макроса. Он включает часть SFC диаграммы, которая таким образом невидима в главном окне редактора. Поток выполнения не изменяется использованием макросов; это просто способ скрыть некоторые части программы, например, с целью упростить отображение. Для вставки блока макроса выполните команду **Insert macro (after)**. Имя макроса может редактироваться.

Для открытия редактора macro дважды нажмите блок macro или выполните команду **Zoom into macro**. Вы можете редактировать здесь так же, как в виде главного редактора и ввести желаемую секцию SFC диаграммы. Чтобы выйти, выполните команду **Zoom out of macro**.

Линия заголовка редактора макроса показывает часть macro внутри текущего SFC примера:

Спецификаторы для действий в SFC

Обзор

С целью конфигурации того, каким путём действия *(см. стр. 331)* должны быть связаны с IEC шагами, доступны некоторые спецификаторы, которые вставляются в поле спецификатора (квалификатора) элемента действия.

Доступные Qualifiers

Qualifier (Специфика-тор)	Длиная форма	Описание
N	non-stored (не сохраняемое)	Действие активно только пока активен шаг.
R0	overriding reset перезаписываемый сброс	Действие становится деактивированным
S0	set (stored) установка	Действие стартует, когда шаг становится активным и будет продолжаться после деактивации шага до акции сброса (reset)
L	time limited ограниченное по времени	Действие стартует, когда шаг становится активным. Оно будет продолжаться, пока шаг не станет неактивным или не пройдёт установленного времени.
D	time delayed задержка по времени	Таймер задержки стартует, когда шаг становится активным. Если шаг всё ещё активен после задержки времени, действие стартует и продолжается, пока не деактивируется.
P	pulse импульс	Действие стартует, когда шаг становится активным / деактивированным и будет выполняться один раз.
SD	stored and time delayed сохраняемое и с задержкой по времени	Действие стартует после установленной задержки времени и будет продолжаться до сброса.
DS	delayed and stored с задержкой и сохранением	Если шаг всё ещё активен после установленной временной задержки, действие стартует и продолжается до сброса.
SL	stored and time limited сохраняемое и ограниченное по времени	Действие стартует, когда шаг становится активным и будет продолжаться для заданного времени или до сброса.

Спецификаторы L, D, SD, DS и SL требуют значения времени в формате константы TIME.

ПРИМЕЧАНИЕ: Когда IEC действие деактивируется, оно будет выполнено еще один раз. Последствие в том, что каждое действие будет выполняться, по крайней мере, дважды.

Неявные переменные - SFC флаги

Обзор

Каждый SFC шаг и IEC действие производит генерируемые неявно переменные для наблюдения за статусом *(см. стр. 339)* шагов и IEC действий во время выполнения. Также Вы можете определять переменные для наблюдения и управления выполнением SFC (тайм-ауты, сброс, tip режим). Эти переменные также могут быть сгенерированы неявно SFC объектом.

Обычно для каждого шага и каждого IEC действия генерируется неявная переменная. Экземпляр структуры, именуемый для элемента, например, step1 для шага с именем **step1**. Вы можете определить в свойствах элемента *(см. стр. 325)* должно ли для этого флага определение символа быть экспортировано в конфигурацию символа и как этот символ должен быть доступным в контроллере.

Типы данных для этих неявных переменных определены в библиотеке IecSFC.library. Эта библиотека автоматически включается в проект, как только добавлен SFC объект.

Статус шага и действия и время шага

Обычно для каждого шага и каждого IEC действия создаётся неявная структурная переменная типа `SFCStepType` или `SFCActionType`. Компоненты структуры (флаги) описывают статус шага или акции или текущее время выполнения активного шага.

Синтаксис декларации для неявно созданной переменной следующий:

<имя шага>: SFCStepType;
или
_<имя действия>:SFCActionType;

ПРИМЕЧАНИЕ: В отличие от предыдущей версии SoMachine неявные переменные для действий присутствуют в V4.0 и позже.

Доступны следующие флаги состояния для шага или действий:

Логические шаги для **шагов**:

Логический флаг	Описание
<stepname>.x	показывает текущий статус активации
<stepname>._x	показывает статус для следующего цикла

Если <stepname>.x = TRUE, шаг будет выполняться в текущем цикле.

Если <stepname>._x = TRUE и <stepname>.x = FALSE, шаг будет выполняться в последующем цикле. Это означает, что <stepname>._x копируется <stepname>.x в начале цикла.

Логические флаги для **действий**:

Логический флаг	Описание
_<actionname>.x	TRUE, если действие выполняется
_<actionname>._x	TRUE, если действие активно

Генерация символов

В свойствах элемента *(см. стр. 325)* шага или действия вы можете определить, если определение символа должно быть добавлено к возможно созданной и загруженной конфигурации символа имени флага для шага или действия.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДУСМОТРЕННОЕ ДЕЙСТВИЕ ОБОРУДОВАНИЯ

Если Вы используете логический флаг <stepname>.x для форсирования определённого статуса шага (для установки шага активным), будьте уверены, что это не приведёт к неконтролируемым состояниям внутри SFC.

Отказ следовать этим указаниям может привести к смерти, серьёзным травмам или повреждению оборудования.

Время через переменные TIME:

Флаг `t` даёт промежуток текущего времени, который прошел после того, так как шаг стал активным. Это только для шагов, неважно есть ли минимальное время, сконфигурированное в признаках шага *(см. стр. 325)* или нет (см. ниже: `SFCError`).

Для **шагов**:
<stepname>.t (<stepname>._t не применим для внешних целей)

Для **действий**:
Неявные переменные времени не используются.

Управление выполнением SFC (Timeouts, Reset, Tip Mode)

Вы можете использовать некоторые неявные доступные переменные, также именуемые SFC флагами (см. таблицу ниже) для управления выполнением SFC. Например, для индикации переполнения тайм-аутов или разрешения tip режима для переключения переходов.

С целью быть способными в доступе к этим флагам Вы должны декларировать и активировать их. Проделайте это в диалоговом окне **SFC Settings**. Это дополнительное окно объекта в диалоговом окне **Properties**.

Ручная декларация, как это было необходимо в SoMachine V3.1, нужна только для разрешения доступа по записи из другой POU (обращайтесь к параграфу *Флаги доступа*).

В этом случае рассмотрите следующее:

Если Вы декларировали флаг глобально, Вы должны деактивировать опцию **Declare** в диалоговом окне **SFC Settings**. Иначе это приведёт к неявно декларированному локальному флагу, который затем будет использоваться вместо глобального. Обратите внимание, что уставки SFC для SFC POU первоначально определяются набором определений в диалоговом окне **Options** → **SFC**.

Примите во внимание, что декларация переменной флага, сделанная исключительно через диалоговое окно SFC Settings, будет видима только в онлайн виде SFC POU.

Могут использоваться следующие неявные переменные (флаги). Для этой цели Вы должны декларировать и активировать их в диалоговом окне **SFC Settings**.

Переменная	Тип	Описание
`SFCInit`	BOOL	Если эта переменная становится равной TRUE, последовательная функциональная диаграмма будет установлена на шаг Init *(см. стр. 327)*. Все шаги и действия и другие SFC флаги будут сброшены (инициализация). Начальный шаг будет оставаться активным, но не будет выполняться, пока переменная равна TRUE. Установите назад `SFCInit` на FALSE, чтобы вернуться к нормальному выполнению.
`SFCReset`	BOOL	Действие этой переменной похоже на `SFCInit`. В отличие от последней, однако, последующая обработка происходит после инициализации начального шага. Следовательно, в этом случае сброс в FALSE флага `SFCReset` может быть сделан в начальном шаге.
`SFCError`	BOOL	Как только какой-либо тайм-аут наступает на одном из шагов в SFC, эта переменная будет установлена в TRUE. Предустановка: `SFCEnableLimit` должен быть TRUE. Примите во внимание, что любой другой тайм-аут не может быть зарегистрирован, прежде чем `SFCError` будет сброшен. `SFCError` должен быть определён. Если вы хотите, используйте другие контролирующие время флаги (`SFCErrorStep`, `SFCErrorPOU`, `SFCQuitError`).
`SFCEnableLimit`	BOOL	Вы можете использовать эту переменную для внешней активации (TRUE) и деактивации (FALSE) контроля времени в шагах через `SFCError`. Это значит, что если эта переменная объявлена и активирована (**SFC Settings**), тогда она должна быть установлена в TRUE, чтобы получить работающий флаг `SFCError`. Иначе, какие-либо тайм-ауты шагов не будут регистрироваться. Использование может быть оправдано во время запуска или при ручном действии. Если переменная не определена, `SFCError` будет работать автоматически. Предустановка: `SFCError` должна быть определена.
`SFCErrorStep`	STRING	Эта переменная хранит имя шага, на котором таймаут был зарегистрирован с помощью `SFCError.timeout`. Предустановка: `SFCError` должна быть определена.
`SFCErrorPOU`	STRING	Эта переменная хранит имя SFC POU, в которой наступил тайм-аут. Предустановка: `SFCError` должна быть определена
`SFCQuitError`	BOOL	Пока эта переменная TRUE, выполнение SFC диаграммы останавливается, и переменная `SFCError` будет сброшена. Когда эта переменная будет сброшена в FALSE, будут сброшены все текущие состояния времени в активных шагах. Предустановка: `SFCError` должна быть определена
`SFCPause`	BOOL	Пока эта переменная TRUE, выполнение SFC диаграммы останавливается.
`SFCTrans`	BOOL	Эта переменная становится TRUE, как только переход приводится в действие.

Переменная	Тип	Описание
SFCCurrentStep	STRING	Эта переменная хранит имя текущего активного шага, независимо от времени мониторинга. В случае одновременных последовательностей будет зарегистрировано имя внешнего правого шага.
SFCTipSFCTipMode	BOOL	Эта переменная позволяет использовать медленный способ выполнения в рамках текущей диаграммы. Когда этот способ был включен SFCTipMode = TRUE, Вы можете только перейти к следующему шагу, установив SFCTip = TRUE (возрастающий фронт). Когда SFCTipMode установлен в FALSE, возможен пропуск с помощью переходов.

Следующая иллюстрация даёт пример некоторых SFC флагов обнаруженных ошибок в онлайн режиме редактора.

Тайм-аут был обнаружен в шаге s1 в SFC объекте **POU** флагом SFCError.

Доступ к флагам

Для разрешения доступа к флагам для управления выполнением SFC (таймауты, перезапуск, tip режим) декларируйте и активируйте переменные флагов, как описано выше (Управление выполнением SFC (см. стр. 340)).

Синтаксис для доступа из действия или перехода внутри SFC POU:
<stepname>.<flag>
или
_<actionname>.<flag>
Примеры:

```
status:=step1._x;
```

```
checkerror:=SFCerror;
```

Синтаксис для доступа из других POU:
<SFC POU>.<stepname>.<flag>
или
<SFC POU>_<actionname>.<flag>
Примеры:

```
status:=SFC_prog.step1._x;
```

```
checkerror:=SFC_prog.SFCerror;
```

Обратите внимание на следующее в случае доступа по записи из другого POU:

- Неявная переменная дополнительно должна быть декларирована явно как VAR_INPUT переменная SFC POU
- или она должна быть декларирована глобально в GVL (global variable list).

Пример: Локальная декларация

```
PROGRAM SFC_prog
VAR_INPUT
  SFCinit:BOOL;
END_VAR
```

Пример: Глобальная декларация в GVL

```
VAR_GLOBAL
  SFCinit:BOOL; END_VAR
```

Доступ к флагу в `PLC_PRG`:

```
PROGRAM PLC_PRG
VAR
  setinit: BOOL;
END_VAR
```

```
SFC_prog.SFCinit:=setinit;          //Write access to SFCinit in SFC_prog
```

Последовательность выполнения в SFC

Обзор

В онлайн режиме конкретные типы действий будут выполняться согласно определённой последовательности; смотрите таблицу ниже.

Определение терминов

Используются следующие термины:

Термин	Описание
active step (активный шаг)	Шаг, чьё действие шага выполняется. В онлайн режиме активные шаги заполняются голубым цветом.
initial step (начальный шаг)	В первом цикле после вызова SFC POU начальный шаг автоматически становится активным, и ассоциированное действие *(см. стр. 331)* выполняется.
IEC actions (IEC действия)	IEC действия выполняются, по крайней мере, дважды: ● Первый раз, когда они становятся активными. ● Второй раз – в следующем цикле – после их деактивации.
alternative branches (альтернативное ветвление)	Если шаг, предшествующий горизонтальной линии начала альтернативного ветвления, активен, тогда первый переход каждой альтернативной ветви будет анализироваться слева направо. Первый переход слева, чьё условие перехода равно TRUE, будет открыт, и последующие шаги будут активированы.
parallel branches (параллельное ветвление)	Если двойная линия в начале линии параллельного ветвления активна и условие предшествующего перехода имеет значение TRUE, тогда во всех параллельных ветвлениях каждый первый шаг становится активным. Ветви теперь будут выполняться параллельно друг другу. Шаг, следующий за двойной линией, в конце ветвления будет активным, когда все предыдущие шаги активны и условие перехода после двойной линии имеет значение TRUE.

Порядок выполнения

Порядок выполнения элементов в последовательности:

Шаг	Описание
1. Сброс IEC действий	Все флаги управления действиями IEC действий *(см. стр. 331)* получают сброс (однако, не флаги IEC действий, которые вызваны внутри действий).
2. Действия выхода из шагов (деактивация шагов)	Все шаги проверяются в порядке, в котором они предполагаются в последовательности (сверху вниз и справа налево), чтобы определить обеспечивается ли условие для исполнения действия выхода из шага. В этом случае оно будет выполнено. Действие выхода будет выполняться, если шаг становится деактивированным *(см. стр. 327)*. Это означает, что если запись и действия шага – если существуют – выполнялись во время последнего цикла и если условие для перехода к следующему шагу равно TRUE.
3. Действия при входе в шаг (активация шага)	Все шаги проверяются в порядке, в котором они предполагаются в последовательности, чтобы определить обеспечивается ли условие для исполнения. В этом случае оно будет выполнено. Записи действий будут выполняться, если предыдущее условие перехода равно TRUE и, следовательно, шаг активируется.
4. Поверка таймаута, действия активного шага	Для не-IEC шагов соответствующие действия активного шага сейчас выполняются в порядке, в котором они позиционированы в последовательности (сверху вниз и справа налево).
5. IEC действия	IEC действия *(см. стр. 331)*, которые используются в последовательности, выполняются в алфавитном порядке. Это делается в 2 проходах через список действий. В первом проходе все IEC действия, которые деактивированы в текущем цикле, выполняются. Во втором проходе все IEC действия, которые активны в текущем цикле, выполняются.
6. Проверка переходов, активация следующих шагов	Переходы *(см. стр. 327)* оцениваются. Если шаг в текущем цикле был активным и следующий переход возвращает TRUE (и если соответствующее минимальное активное время уже прошло), тогда следующий шаг активируется

ПРИМЕЧАНИЕ: Действие может быть выполнено несколько раз в одном цикле, потому что оно вызывается из более чем одного IEC действия, когда есть несколько активных шагов. Другими словами, одно и то же IEC действие используется одновременно в различных уровнях SFC, и это может привести к непредусмотренным эффектам.

Пример: У SFC могло быть 2 действия IEC A и B, которые оба реализованы в SFC, и которые оба вызывают IEC действия C. Тогда в действиях IEC A и B и могут быть активными в том же самом цикле и, следовательно, в обоих IEC действиях действие C может быть активным. Тогда C было бы вызвано дважды.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДУСМОТРЕННОЕ ДЕЙСТВИЕ ОБОРУДОВАНИЯ

Не вызывайте IEC действия из нескольких других IEC действий в одном и том же цикле.

Отказ следовать этим указаниям может привести к смерти, серьёзным травмам или повреждению оборудования.

ПРИМЕЧАНИЕ: Используйте неявные переменные *(см. стр. 339)* для определения статуса шагов, действий или выполнения диаграммы.

SFC редактор в онлайн режиме

Обзор

В онлайн режиме SFC редактор обеспечивает виды для мониторинга (смотрите ниже) и для записи и форсирования переменных и выражений в контроллере. Функциональность отладки как для других IEC языков (точки останова, пошаговое выполнение и т.п.) невозможна в SFC. Однако, рассмотрите следующие подсказки для отладки SFC:

- За информацией о том, как открыть объект в режиме онлайн обращайтесь к описанию интерфейса пользователя в онлайн режиме *(см. стр. 49)*.
- Окно редактора SFC объекта также включает редактор деклараций в верхней части. За дальнейшей информацией обращайтесь к главе *Редактор деклараций в режиме онлайн (см. стр. 376)*. Если Вы имеете декларированные неявные переменные (SFC флаги) *(см. стр. 339)* через диалоговое окно SFC **Settings**, они будут добавлены здесь, но не будут показаны в режиме офлайн редактора деклараций.
- Рассмотрите последовательность выполнения *(см. стр. 344)* элементов Sequential Function Chart.
- Просмотрите свойства объекта или опции SFC редактора и SFC значения по умолчанию, касающиеся компиляции или отображения онлайн элементов SFC и их атрибутов.
- Рассмотрите возможное использование флагов *(см. стр. 339)* для наблюдения и управления выполнением.

Мониторинг

Активные шаги отображаются заполненными голубым цветом. Отображение атрибутов зависит от установки опций SFC редактора.

Онлайн вид объекта программы `SFC_prog`

Глава 13

Редактор Structured Text (ST)

Содержание этой главы

Эта глава содержит следующие элементы:

Раздел	Тема	Страница
13.1	Информация о ST редакторе	348
13.2	Язык Structured Text ST / Extended Structured Text (ExST) (структурированного текста / расширенного структурированного текста)	354

Раздел 13.1

Информация о ST редакторе

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
ST редактор	349
ST редактор в онлайн режиме	350

ST редактор

Обзор

ST редактор используется для создания программных объектов в языке программирования IEC Structured Text (ST) – Структурированный текст – или extended Structured Text, который обеспечивает некоторые расширения стандарта IEC 61131-3.

ST редактор – это текстовый редактор. Следовательно, используйте соответствующие уставки в диалоговых окнах **Options** и **Customize**, чтобы сконфигурировать поведение, вид и меню. Там Вы можете определить уставки по умолчанию для выделения цветом, числа строк, табуляции, отступы и много других опций.

Дальнейшая информация

Чтобы выбрать блоки, нажмите клавишу ALT и выберите желаемую область текста с помощью мыши.

Редактор будет доступен в нижней части окна, которое также включает редактор деклараций (объявлений) *(см. стр. 372)* в верхней части.

Если во время редактирования обнаруживаются синтаксические ошибки, соответствующие сообщения будут отображаться в окне **Precompile Messages**. Это окно обновляется каждый раз, когда Вы переустанавливаете фокус в окно редактора (например, помещаете Ваш курсор в другое окно и затем возвращаете назад в окно редактора.)

ST редактор в онлайн режиме

Обзор

В онлайн режиме редактор структурированного текста (ST editor) обеспечивает виды для мониторинга *(см. стр. 350)*, записи и форсирования переменных и выражений в контроллере. Отладка (точки останова, пошаговое выполнение и т.п.) доступна. Смотрите Позиции остановки в редакторе ST *(см. стр. 352)*.

- За информацией о том, как открыть объекты в онлайн режиме, обращайтесь к описанию интерфейса пользователя в онлайн режиме *(см. стр. 49)*.
- За информацией о том, как ввести подготовленные значения для переменных в онлайн режиме, смотрите Форсирование переменных *(см. стр. 351)*.
- Окно редактора ST объекта также включает редактор деклараций в верхней части. За информацией о редакторе деклараций в онлайн режиме смотрите Редактор деклараций в онлайн режиме *(см. стр. 376)*.

Мониторинг

Если мониторинг явно не деактивирован в диалоговом окне **Options**, маленькие блоки мониторинга будут отображаться за каждой переменной, показывая актуальное значение.

Онлайн вид объекта программы `PLC_PRG` с мониторингом:

Форсирование переменных

В добавление к возможности ввода подготовленного для переменной значения внутри декларации любого редактора, ST редактор предлагает двойное нажатие блока мониторинга переменной в части реализации (в онлайн режиме). Вводите подготовленное значение в возникающем диалоговом окне.

Диалоговое окно **Prepare Value**

Вы найдёте имя переменной, дополненное её путём в **Devices Tree** (**Expression**), её тип и текущее значение.
Активацией соответствующего варианта Вы можете выбрать следующие опции:

- подготовка нового значения, которое должно быть введено в поле редактирования
- удаление подготовленного значения
- реализация переменной, которая форсированно устанавливается
- реализация переменной, которая была форсированно установлена, и переустановка её на значение, которое она имела до форсирования

Для выполнения выбранного действия выполните команду **Debug** → **Force values** (тема **Online**) или нажмите клавишу F7.

Позиции точек останова в ST редакторе

Вы можете настроить точку останова, как обычно, в позициях POU, где значения переменных могут изменяться, или где программный поток разветвляется, или вызывается другая POU. В последующих описаниях `{BP}` указываются возможные позиции точки останова (breakpoint).

Назначение:

В начале линии. Обратите внимание, что это назначение, как выражения *(см. стр. 356)*, определяет отсутствие дальнейших позиций точек останова внутри линии.

- **FOR**-цикл:

1. перед инициализацией счётчика
2. перед тестированием счётчика
3. перед выражением

```
{BP} FOR i := 12 TO {BP} x {BP} BY 1 DO
{BP} [statement1]
...
{BP} [statementn-2]
END_FOR
```

- **WHILE**-цикл:

1. перед проверкой условия
2. перед инструкцией

```
{BP} WHILE i < 12 DO
{BP} [statement1]
...
{BP} [statementn-1] END_WHILE
```

REPEAT-цикл:

- перед проверкой условия

```
REPEAT
{BP} [statement1]
...
{BP} [statementn-1]
{BP} UNTIL i >= 12 END_REPEAT
```

Вызов программы или функционального блока:

В начале линии.

```
{{BP} POU( );
```

В конце POU:

При выполнении по шагам эта позиция будет также достигнута после инструкции `RETURN`.

Отображение точек остановок в ST

Breakpoint в онлайн режиме	Заблокированная Breakpoint	Остановка программы в Breakpoint

ПРИМЕЧАНИЕ: Точка останова будет установлена автоматически во всех методах, которые могут быть вызваны. Если вызывается метод, управляемый интерфейсом, точки останова будут установлены во всех методах функциональных блоков, реализующих этот интерфейс, и во всех производных функциональных блоках, подписанных на метод. Если метод вызывается через указатель на функциональный блок, точки останова будут установлены в методе функционального блока и во всех производных функциональных блоках, подписанных на метод.

Раздел 13.2

Язык Structured Text ST / Extended Structured Text (ExST)

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Структурированный текст ST / Расширенный структурированный текст ExST	355
Выражения	356
Инструкции	358

Структурированный текст ST / Расширенный структурированный текст ExST

Обзор

Structured Text – текстовый язык программирования высокого уровня, аналогичный PASCAL или C. Программный код составляется из выражений *(см. стр. 356)* и инструкций *(см. стр. 358)*. В отличие от IL (Instruction List), Вы можете использовать многочисленные конструкции для программирования циклов, таким образом разрешая разработку сложных алгоритмов.

Пример

```
IF value < 7 THEN
 WHILE value < 8 DO
  value:=value+1;
 END_WHILE;
END_IF;
```

Extended Structured Text (ExST) – специальное расширение SoMachine стандарта IEC 61131-3 для Structured Text (ST). Примеры: назначения как выражения, операторы set/reset.

Выражения

Обзор

Выражение – это конструкция, которая после своего вычисления возвращает значение. Это значение используется в инструкциях.

Выражения составляются из операторов *(см. стр. 617)*, операндов *(см. стр. 709)*, и/или назначений. Операнд может быть константой, переменной, вызовом функции или другим выражением.

Примеры

`33`	(* Константа *)
`ivar`	(* Переменная *)
`fct(a,b,c)`	(* Вызов функции *)
`a AND b`	(* Выражение *)
`(x*y) / z`	(* Выражение *)
`real_var2 := int_var;`	(* Назначение, см. ниже*)

Порядок операций

Вычисление выражения производится выполнением операторов, согласно определённым правилам. Оператор с наивысшим уровнем операции выполняется первым, затем оператор со следующим уровнем операции и так далее, пока не будут выполнены все.

Ниже Вы найдёте таблицу ST операторов в порядке их операционного уровня:

Операция	Символ	Операционный уровень
помещение в скобки	(выражение)	высший приоритет
вызов функции	имя функции (список параметров)	.....
возведение в степень	EXPT	.....
отрицание	–	.....
создание дополнения	NOT	.....
умножение	*	.....
деление	/	.....
модуль	MOD	.....
сложение	+	.....
вычитание	–	.....
сравнение V	<,>,<=,>=	.....
равенство	=	....
неравенство	<>	...
логическое AND	AND	..

Операция	Символ	Операционный уровень
логическое XOR	XOR	.
логическое OR	OR	низший приоритет

Назначение выражений

В расширении стандарта IEC 61131-3 (ExST), назначения могут быть использованы как выражения.

Примеры:

`int_var1 := int_var2 := int_var3 + 9;`	(* `int_var1` и `int_var2` оба равны значению `int_var3` + 9*)
`real_var1 := real_var2 := int_var;`	(* верные назначения, `real_var1` и `real_var2` получат значения `int_var` *)
`int_var := real_var1 := int_var;`	(* будет отображено сообщение из-за несоответствия типов `real-int` *)

`IF b := (i = 1) THEN`
`i := i + 1;`
`END_IF`
	(*Выражение используется внутри выражения условия IF: Сначала `b` будет назначено TRUE или FALSE, от того `i` равно 1 или нет, затем будет вычислено результирующее значение `b`.*)

Инструкции

Обзор

Инструкции описывают то, что делается с данными выражениями *(см. стр. 356)*.

В ST могут быть использованы следующие инструкции:

Инструкция	Пример
назначение *(см. стр. 359)*	`A:=B; CV := CV + 1; C:=SIN(X);`
Вызов функционального блока *(см. стр. 360)* и использование выхода функционального блока	`CMD_TMR(IN := %IX5, PT := 300);` `A:=CMD_TMR.Q`
`RETURN` *(см. стр. 360)*	`RETURN;`

`IF` *(см. стр. 361)*	`D:=B*B;`
	`IF D<0.0 THEN`
	`C:=A;`
	`ELSIF D=0.0 THEN`
	`C:=B;`
	`ELSE`
	`C:=D; END_IF;`

`CASE` *(см. стр. 362)*	`CASE INT1 OF`
	`1: BOOL1 := TRUE;`
	`2: BOOL2 := TRUE; ELSE`
	`BOOL1 := FALSE;`
	`BOOL2 := FALSE;`
	`END_CASE;`

`FOR` *(см. стр. 363)*	`J:=101;`
	`FOR I:=1 TO 100 BY 2 DO`
	`IF ARR[I] = 70 THEN`
	`J:=I;`
	`'EXIT; END_IF; END_FOR;`

`WHILE` *(см. стр. 364)*	`J:=1;`
	`WHILE J<= 100 AND ARR[J] <> 70 DO`
	`J:=J+2;`
	`END_WHILE;`

`REPEAT` *(см. стр. 365)*	`J:=-1;`
	`REPEAT`
	`J:=J+2;`
	`UNTIL J= 101 OR ARR[J] = 70`
	`END_REPEAT;`

`EXIT` *(см. стр. 366)*	`EXIT;`

Инструкция	Пример
CONTINUE *(см. стр. 366)*	CONTINUE;

JMP *(см. стр. 366)*	label1: i:=i+1;
	IF i=10 THEN
	  JMP label2;
	END_IF
	JMP label1;
	label2:

пустая инструкция	;

Операторы назначения

Стандартное назначение

В левой части назначения есть операнд (переменная, адрес) которому значение выражения в правой части назначается оператором назначения `:=`.

Также обращайтесь к описанию оператора `MOVE` *(см. стр. 630)*, который имеет такую же функцию.

Пример

```
Var1 := Var2 * 10;
```

После завершения этой линии `Var1` имеет десятикратное значение `Var2`.

Set оператор `S=`

Значение будет установлено: если однажды установлено в TRUE, оно будет оставаться TRUE.

Пример

```
a S= b;
```

`a` получает значение `b`: если однажды установлено в TRUE, оно будет оставаться TRUE даже если `b` станет снова FALSE.

Reset оператор `R=`

Значение будет сброшено: если однажды установлено в FALSE, оно будет оставаться FALSE.

Пример

```
a R= b;
```

`a` устанавливается в FALSE пока `b` = TRUE.

ПРИМЕЧАНИЕ: В случае многократного назначения, set и reset назначений обращайтесь к последнему члену назначения.

Пример

```
a S= b R= fun1(par1,par2)
```

В этом случае `b` будет сброшенным значением `fun1`. Но `a` не получит значение установки `b`, оно получит значение установки выходного значения `fun1`.

Примите во внимание, что назначение может быть использовано как выражение *(см. стр. 356)*. Это расширение стандарта IEC 61131-3.

Вызов функциональных блоков в ST

Функциональный блок (сокращенно как FB) вызывается в структурированном тексте согласно следующему синтаксису:

<имя экземпляра FB >(входная переменная FB:=<значение или адрес>|, <дальнейшая входная переменная FB:=< значение или адрес >|...дальнейшие входные переменные FB);

Пример

В следующем примере функциональный блок таймера (`TON`) вызывается с назначением параметров `IN` и `PT`. Затем результирующая переменная `Q` назначается переменной `A`. Функциональный блок устанавливается на `TMR:TON;`. Результирующая переменная, как в IL, согласно синтаксису <FB instance name>.<FB variable>:

```
TMR(IN := %IX5, PT := 300);
```

```
A:=TMR.Q;
```

Есть также другой синтаксис, доступный для выходов:

```
fb(in1:=myvar, out1=>myvar2);
```

Инструкция RETURN

Вы можете использовать инструкцию `RETURN` для выхода из POU.

Синтаксис

```
RETURN;
```

Пример

```
IF b=TRUE THEN
RETURN;
END_IF;
a:=a+1;
```

Если `b` TRUE, инструкция `a:=a+1;` не будет выполняться. Как результат, POU будет немедленно покинута.

Инструкция IF

С инструкцией IF Вы можете тестировать условие и, в зависимости от этого условия, выполнять инструкции.

Синтаксис

IF <boolean_expression1> THEN

<IF_instructions>

{ELSIF <boolean_expression2> THEN

<ELSIF_instructions1>

..

..

ELSIF <boolean_expression n> THEN

<ELSIF_instructions-1>

ELSE

<ELSE_instructions>}

END_IF;

Сегмент в скобках {} является опционным.

Если <boolean_expression1> возвращает TRUE, тогда выполняются только <IF_instructions> и как результат, не какие-либо другие инструкции. Иначе логические выражения, начинающиеся<boolean_expression2>, вычисляются одна за другой, пока первое выражение возвращает TRUE. Затем вычисляются только те инструкции после этого логического выражения и перед следующим ELSE или ELSIF. Если нет логических выражений, дающих TRUE, тогда вычисляются только <ELSE_instructions>.

Пример

```
IF temp<17
THEN heating_on := TRUE;
ELSE heating_on := FALSE;
END_IF;
```

Здесь нагревание включается, когда температура опускается ниже 17 градусов. Иначе оно остаётся отключённым.

Инструкция CASE

С инструкцией CASE вы можете комбинировать несколько последовательных инструкций с условной переменной в одну конструкцию.

Синтаксис

CASE <Var1> OF

<value1>: <instruction 1>

<value2>: <instruction 2>

<value3, value4, value5>: <instruction 3>

<value6..value10>: <instruction 4>

..

..

<value n>: <instruction n>

ELSE <ELSE instruction>

END_CASE;

Инструкция CASE выполняется согласно следующей модели:

- Если переменная в <Var1> имеет значение <Value I>, тогда будут выполнены инструкции <Instruction I>.
- Если <Var 1> не имеет указанных значений, тогда будут выполняться <ELSE Instruction>.
- Если одна и та же инструкция должна выполняться для нескольких значений переменной, тогда Вы можете записать эти значения одно за другим, разделённые запятыми, и, следовательно, условие общего выполнения.
- Если одна и та же инструкция должна выполняться для диапазона значений переменной, Вы можете записать начальное значение и конечное значение, разделенные двоеточием. Следовательно, Вы можете задать общее условие.

Пример

```
CASE INT1 OF
1, 5: BOOL1 := TRUE;
   BOOL3 := FALSE;
2: BOOL2 := FALSE;
      BOOL3 := TRUE;
10..20: BOOL1 := TRUE;
      BOOL3:= TRUE;
ELSE
    BOOL1 := NOT BOOL1;
    BOOL2 := BOOL1 OR BOOL2;
END_CASE;
```

Цикл FOR

С циклом FOR Вы можете программировать повторяющиеся процессы.

Синтаксис

INT_Var:INT;

FOR <INT_Var> := <INIT_VALUE> TO <END_VALUE> {BY <step size>} DO

<instructions>

END_FOR;

Сегмент в скобках {} опционный.

<instructions> выполняется пока счётчик <INT_Var> не больше, чем <END_VALUE>. Это проверяется до выполнения <instructions>, поэтому <instructions> не выполняется если <INIT_VALUE> больше, чем <END_VALUE>.

Когда <instructions> выполняются, <INT_Var> увеличивается на <Step size>. Размер шага может иметь любое целое значение. Если он пропущен, тогда он устанавливается на 1. Цикл прекращается, когда <INT_Var> больше чем <END_VALUE>.

Пример

```
FOR Counter:=1 TO 5 BY 1 DO
Var1:=Var1*2;
END_FOR;
Erg:=Var1;
```

Предположим, что значение по умолчанию для Var1 равно 1. Тогда она будет иметь значение 32 после цикла FOR.

ПРИМЕЧАНИЕ: Если равно предельному значению для типа данных <INT_Var> (Counter в примере выше), Вы будете производить неопределённый или бесконечный цикл. Если, например, Counter типа SINT и <END_VALUE> равно 127 (максимальное положительное значение для переменной типа SINT), тогда цикл может никогда не прерваться, потому что добавление 1 к этому максимальному значению дало бы в результате, что переменная становится негативной и некогда не превысит предел, заданный инструкцией FOR.

⚠ ПРЕДУПРЕЖДЕНИЕ

БЕСКОНЕЧНЫЙ ЦИКЛ, ПРИВОДЯЩИЙ К НЕПРЕДУСМОТРЕННОМУ ДЕЙСТВИЮ ОБОРУДОВАНИЯ

Гарантируйте, что тип переменной, используемой в инструкциях FOR, имеет достаточную емкость (достаточно больше верхнего предела) для подсчёта <END_VALUE> + 1.

Отказ следовать этим указаниям может привести к смерти, серьёзным травмам или повреждению оборудования.

Вы можете использовать инструкцию CONTINUE внутри цикла FOR. Это является расширением стандарта IEC 61131-3.

Цикл WHILE

Альтернативой для цикла FOR является цикл WHILE, который выполняет цикл если и пока логическое условие является и остаётся равным TRUE. Если условие вначале не TRUE, цикл не выполняется. Если условие, которое вначале было TRUE, становится FALSE, цикл прерывается.

Синтаксис

WHILE <boolean expression> DO

<instructions>

END_WHILE;

Очевидно, начальное и продолжающееся булево выражение должно принять значение FALSE в какой-то момент в рамках инструкций цикла. Иначе цикл не прекратится, что приведет к неопределённому или бесконечному циклу выполнения.

⚠ ПРЕДУПРЕЖДЕНИЕ

БЕСКОНЕЧНЫЙ ЦИКЛ, ПРИВОДЯЩИЙ К НЕПРЕДУСМОТРЕННОМУ ДЕЙСТВИЮ ОБОРУДОВАНИЯ

Гарантируйте, что цикл REPEAT будет прерван внутри цикла инструкций созданием условия FALSE логического выражения.

Отказ следовать этим указаниям может привести к смерти, серьёзным травмам или повреждению оборудования.

Последующее является примером инструкций в цикле, вызывающих прерывание цикла:

```
WHILE Counter<>0 DO
   Var1 := Var1*2;
   Counter := Counter-1;
END_WHILE
```

Инструкция REPEAT еще не была представлена, потому перемещайтесь к параграфу (с модификацией) ниже.

Вы можете использовать инструкцию CONTINUE внутри цикла WHILE.

Цикл REPEAT

Цикл REPEAT - другая альтернатива для цикла FOR, как и для цикла WHILE. Цикл REPEAT отличается от цикла WHILE тем, что условие выхода вычисляется только после выполнения цикла, по крайней мере, однажды, в конце цикла.

Синтаксис

REPEAT

<instructions>

UNTIL <boolean expression>

END_REPEAT;

<instructions> выполняются повторно, пока <boolean expression> остаётся TRUE. Если <boolean expression> производится уже в первом вычислении UNTIL, тогда <instructions> выполняются только однажды. <boolean expression> должны принимать значение TRUE в некоторой точке внутри цикла инструкций. Иначе цикл не прервётся, приводя к неопределённому или бесконечному условию цикла.

⚠ ПРЕДУПРЕЖДЕНИЕ

БЕСКОНЕЧНЫЙ ЦИКЛ, ПРИВОДЯЩИЙ К НЕПРЕДУСМОТРЕННОМУ ДЕЙСТВИЮ ОБОРУДОВАНИЯ

Гарантируйте, что цикл REPEAT будет прерван внутри цикла инструкций созданием условия TRUE логического выражения.

Отказ следовать этим указаниям может привести к смерти, серьёзным травмам или повреждению оборудования.

Последующее является примером инструкций в цикле, вызывающем его прерывание:

```
REPEAT
 Var1 := Var1*2; Counter :=
 Counter-1;
UNTIL
       Counter=0
      END_REPEAT;
```

Вы можете использовать инструкцию CONTINUE внутри цикла REPEAT. Это является расширением стандарта IEC 61131-3.

Циклы WHILE и REPEAT в определённом смысле более мощные, чем цикл FOR, потому что Вы не должны знать число циклов до выполнения цикла. В некоторых случаях Вы сможете, следовательно, способны, работать с этими двумя типами циклов. Если, однако, число циклов ясно, тогда цикл FOR предпочтительней, так как в большинстве случаев это неотъемлемо исключает бесконечные петли (см. предупреждающее сообщение параграфа для цикла FOR (*см. стр. 363*)).

Инструкция CONTINUE

Как расширение IEC 61131-3 стандарта инструкция CONTINUE поддерживается внутри циклов FOR, WHILE, и REPEAT. CONTINUE заставляет перейти к началу следующего цикла выполнения.

Пример

```
FOR Counter:=1 TO 5 BY 1 DO
INT1:=INT1/2;
IF INT1=0 THEN
CONTINUE; (* to avoid division by zero *)
END_IF
Var1:=Var1/INT1; (* only executed, if INT1 is not "0" *)
END_FOR;
Erg:=Var1;
```

Инструкция EXIT

Инструкция EXIT прекращает циклы FOR, WHILE или REPEAT, в котором это происходит без отношения к какому-либо условию.

Инструкция JMP

Вы можете использовать инструкцию JMP для безусловного перехода линии кода помеченной меткой перехода.

Синтаксис

JMP <label>;

<label> - произвольный, но уникальный идентификатор, который помещен в начале линии программы. Инструкция JMP должна сопровождаться указанием места назначения перехода, которое должно равняться предопределенной метке.

⚠ ПРЕДУПРЕЖДЕНИЕ

БЕСКОНЕЧНЫЙ ЦИКЛ, ПРИВОДЯЩИЙ К НЕПРЕДУСМОТРЕННОМУ ДЕЙСТВИЮ ОБОРУДОВАНИЯ

Гарантируйте, что использование инструкции JMP в условиях, таких как это, не приведёт к неопределённому или бесконечному циклу.

Отказ следовать этим указаниям может привести к смерти, серьёзным травмам или повреждению оборудования.

Последующее является примером инструкций в создании логических условий, которые предотвращают неопределённый или бесконечный цикл между инструкцией перехода и её назначением:

```
aaa:=0;
_label1: aaa:=aaa+1;
(*instructions*)
IF (aaa < 10) THEN
JMP _label1;
END_IF;
```

Пока переменная `aaa`, инициализированная значением 0, имеет значение меньше 10, инструкция jump в примере выше будет приводить к повторному переходу линии программы, определённой меткой `_label1`. Следовательно, это будет вызывать повторяющееся выполнение инструкций, расположенных между меткой и инструкцией `JMP`. Так как эти инструкции также включает инкремент переменной `aaa`, условие перехода будет нарушено (в девятой проверке) и процесс выполнения программы продолжится.

Вы можете также получить эту же функциональность при помощи циклов `WHILE` или `REPEAT` в примере. Обычно использование инструкций безусловных переходов уменьшает удобочитаемость кода.

Комментарии в ST

Есть две возможности записать комментарии в объектах структурированного текста:

- Начать комментарий с `(*` и закрыть его `*)`. Это позволяет Вам вставить комментарий, который распространяется на несколько линий. Пример: (*This is a comment.*)
- Комментарии в одной линии как расширение стандарта IEC 61131-3: `//` наклонные линии начинают комментарий, который заканчивается в конце линии. Пример: // This is a comment.

Вы можете помещать комментарии везде внутри декларативной или реализационной части ST редактора.

Вложенные комментарии: Вы можете помещать комментарии в рамках других комментариев.

Пример

```
(*
a:=inst.out; (* to be checked *)
b:=b+1;
*)
```

В этом примере комментарий, который начинается с первой скобки, закрывается не последующей скобкой, следующей за `checked`, а только последней скобкой.

Часть V

Редакторы объектов

Содержание этой части

Эта часть содержит следующие главы:

Глава	Название главы	Страница
14	Редакторы объявлений	371
15	Редактор Device Type Manager (DTM)	377
16	Редактор Data Unit Type (DUT)	379
17	Редактор Global Variables List (GVL)	381
18	Редактор Network Variables List (NVL)	383
19	Редактор задач	405
20	Редактор списка наблюдений	415
21	Инструменты в логических редакторах	421

Глава 14

Редакторы объявлений

Обзор

Текстовый редактор объявлений служит для создания раздела деклараций (объявлений) объекта POU. Он может быть добавлен табличным представлением. Любая модификация, сделанная в одном из представлений, сразу применяется к другому.

В зависимости от текущих параметров настройки опции в редакторе объявлений, будет доступно или только текстовое или только табличное представление. Вы можете переключаться между ними посредством кнопок (**Textual**-Текстовый /**Tabular**- Табличный) на правой границе окна редактора.

Обычно, редактор объявлений используется в сочетании с редакторами языка программирования. Это означает, что он будет помещён в верхнюю часть окна, которое открывается, когда Вы собираетесь отредактировать или просмотреть (проконтролировать) объект в режиме офлайн или режиме онлайн. Заголовок объявления описывает тип POU (например: PROGRAM, FUNCTION_BLOCK, FUNCTION). Он может быть расширен с помощью глобальных атрибутов прагмы POU.

Обращайтесь также к главе *Декларация переменных (см. стр. 497)*.

Режим онлайн редактора объявлений *(см. стр. 376)* структурирован как вид просмотра **Watch**.

Объявление переменной также осуществляется в списках глобальных переменных (**GlobalVariableLists)** и типов блоков данных (**DataUnitTypes)**, которые создаются в отдельных редакторах.

Обратитесь также к главе *Объявление переменных (см. стр. 497)*.

Содержание этой главы

Эта глава содержит следующие темы:

Тема	Страница
Текстовый редактор объявлений	372
Табличный редактор объявлений	373
Редактор объявлений в режиме онлайн	376

Текстовый редактор объявлений

Обзор

Вид текстового редактора

Поведение и видимость определены соответствующими текущими параметрами настройки текстового редактора в диалоговых окнах **Options** и **Customize**. Там Вы можете определить настройки по умолчанию для выделения цветом, числа строк, вкладок, расположения с отступом и еще многих опций. Доступны обычные функции редактирования, такие как копирование и вставка. Выделение блока возможно нажатием клавиши ALT при выборе мышью желаемой текстовой области.

Табличный редактор объявления

Обзор

Вид табличного редактора

Чтобы добавить новую строку объявления выше существующего, сначала выберите эту строку и выполните команду **Insert** на панели инструментов или в контекстном меню.

Чтобы добавить новое объявление в конце таблицы, щелкните вне последней существующей строки объявления и также используйте команду **Insert**.

Новое вставленное объявление по умолчанию сначала использует вид **VAR** и недавно введенный тип данных. Поле ввода для обязательного имени переменной **Name** открывается автоматически. Введите допустимый идентификатор и закройте поле, нажав клавишу ENTER или щелкнув по другой части вида.

Дважды щелкните по ячейке таблицы, чтобы открыть соответствующие возможности для введения значения.

Дважды щелкните по **Scope**, чтобы открыть список, из которого Вы можете выбрать желаемый объем и определить объем ключевого слова атрибута (флаг).

Введите информацию в **Datatype** (тип данных) непосредственно или щелкните кнопку**>**, чтобы использовать **InputAssistant** или **Arraywizard**.

Введите информацию в **Initialization** (инициализация) непосредственно или щелкните кнопку **...**, чтобы открыть диалоговое окно **Initialization value**. Это особенно полезно в случае структурированных переменных.

Каждая переменная объявлена в отдельной строке, где строки пронумерованы.

Вы можете изменить порядок строк (номер строки), выбрав строку, и переместив её вверх или вниз командой **Moveup** или командой **Movedown** на панели инструментов или в контекстном меню.
Вы можете сортировать список объявлений согласно каждому из столбцов, щелкнув по заголовку соответствующего столбца. Столбец, который в настоящее время определяет порядок, обозначен символом стрелки:
стрелка вверх = стрелка порядка по возрастанию
стрелка вниз = порядок по убыванию
Каждый дальнейший щелчок по заголовку столбца изменяет порядок между порядком по возрастанию и порядком по убыванию.

Чтобы удалить одно или несколько объявлений, выберите соответствующие строки и нажмите клавишу DEL или выполните команду **Delete** из контекстного меню или нажмите кнопку на панели инструментов.

Значение при инициализации

Диалоговое окно значений при инициализации **Initialization value**

Выражения **Expressions** переменной выведены на экран с текущими значениями при инициализации. Выберите желаемые переменные и отредактируйте значение при инициализации в поле ниже перечисления. После этого нажмите клавишу **Apply value to selected lines** (применить значения к выделенным линиям). Чтобы восстановить значения инициализации по умолчанию, щелкните кнопку **Reset selected lines to default values** (восстановить выбранные линии до значений по умолчанию).

Нажмите CTRL + ENTER, чтобы вставить разрывы строк в запись **Comment**.

Редактирование заголовка декларации

Вы можете отредактировать заголовок объявления в диалоговом окне **Edit Declaration Header.** Откройте его, щелкнув по панели вверху редактора (PROGRAMPLC_PRG на рисунке выше) или через команду **Edit Declaration Header**.

Диалоговое окно редактирования заголовка декларации **Edit Declaration Header**

Диалоговое окно редактирования заголовка декларации **Edit Declaration Header** содержит следующие элементы:

Элемент	Описание
Declaration	Введите тип (из списка выбора) и имя POU объекта.
Comment	Введите комментарий. Нажмите CTRL + ENTER, чтобы вставить разрывы линии.
Attributes	Открывает диалоговое окно **Attributes** (см. далее ниже в этой главе) для вставки прагм и атрибутов.

Атрибуты

В диалоговом окне **Edit Declaration Header** нажмите кнопку **Attributes...**, чтобы открыть диалоговое окно **Attibutes**. Это позволяет Вам вводить многочисленные атрибуты и прагмы в текстовом формате. Вставляйте их, не заключая фигурные скобки {}, используйте отдельную строку для каждого элемента. Для примера, показанного на следующей иллюстрации, посмотрите соответствующий текстовый вид выше на рисунке текстового вида редактора *(см. стр. 372)*.

Диалоговое окно **Attributes**

Редактор декларации в режиме онлайн

Обзор

После входа в систему контроллера каждый объект, который был уже открыт в окне в режиме офлайн, будет автоматически выведен на экран в онлайновом виде.

Онлайн вид редактора деклараций представляет таблицу, сходную с используемой в виде просмотра *(см. стр. 419)*. Линия заголовка показывает фактический путь до объекта <devicename>.<applicationname>.<objectname>. Таблица для каждого выражения просмотра показывает тип и текущее значение, а также - если в настоящее время установлено – подготовленное значение для форсированного назначения или записи. Если возможно, непосредственно назначенный IEC **Address** и / или **Comment** показаны в дальнейших колонках.

Чтобы установить подготовленное значение для переменной, или используйте диалоговое окно **Prepare Value,** или щелкните по полю назначения столбца **Prepared value** и непосредственно введите желаемое значение. В случае перечислений откроется список, показывающий перечисляемые величины, для выбора значения. В случае логической переменной обработка еще проще.

Вы можете переключать подготовленные булевы значения при помощи клавиши RETURN или SPACE(пробел) в следующем порядке:

- Если значение TRUE, шаги подготовки следующие FALSE -> TRUE -> неизменно.
- Если значение FALSE, шаги подготовки следующие TRUE -> FALSE -> неизменно.

Если наблюдаемое выражение (переменная) является структурированным типом, например, экземпляром функционального блока или переменной типа массив, то знак плюса или минуса предшествуют выражению. При щелчке мышью по этому знаку определенные элементы установленного объекта могут быть дополнительно отображены (см. `fbinst` в последующем изображении), или скрыты (см. `aVar`). Значки указывают, является ли переменная входом , выходом или обычной переменной .

Когда Вы укажете курсором на переменной в части реализации, подсказка покажет декларацию и комментарий переменной. Смотрите следующее изображение, показывающее редактор деклараций в верхней части программного объекта `PLC_PRG` в режиме онлайн:

Вид онлайн редактора декларации

Глава 15

Редактор Device Type Manager (DTM)

Редактор DTM

Обзор

Вид редактора DTM зависит от менеджера типов устройств Device Type Manager.

Изображение показывает пример редактора DTM.

Для получения дополнительной информации по DTM, обращайтесь к *SoMachine Device Type Configuration (DTM) User Guide (см. SoMachine, Device Type Manager (DTM), User Guide)*.

Для списка версий DTM, в настоящее время поддерживаемых SoMachine, обратитесь к Информации о версии Вашей установки SoMachine.

Глава 16

Редактор Data Unit Type (DUT)

Редактор Data Unit Type

Обзор

Вы можете создать определяемые пользователем типы данных *(см. стр. 578)* в редакторе Data Unit Type (редактор DUT). Это - текстовый редактор и он ведет себя согласно установленному в данное время набору опций текстового редактора.

Редактор DUT будет открыт автоматически в окне, когда Вы добавите объект DUT в диалоговом окне **Addobject**. В этом случае обеспечивается по умолчанию синтаксис расширенного описания структуры. Вы можете использовать его, как желаемый для ввода декларации простой структуры или ввода декларации другого типа данных, например перечисления.

Редактор открывается также, когда Вы открываете существующий объект DUT, выбранный в настоящее время в виде POU.

Окно редактора DUT

Глава 17

Редактор Global Variables List (GVL)

Редактор GVL

Обзор

Редактор GVL – **Declaration Editor** для редактирования списков глобальных переменных (**Global Variables Lists)**. Редактор GVL работает, как Редактор объявлений и соответствует опциям, как офлайн, так и онлайн, установленным для текстового редактора. Объявление начинается с `VAR_GLOBAL` и заканчивается `END_VAR`. Эти ключевые слова обеспечиваются автоматически. Введите между ними допустимые объявления глобальных переменных.

Редактор GVL

Глава 18

Редактор Network Variables List (NVL)

Содержание главы

Эта глава содержит следующие разделы:

Раздел	Тема	Страница
18.1	Информация о редакторе NVL	384
18.2	Общая информация о сетевых переменных	385

Раздел 18.1

Информация о редакторе NVL

Редактор списка сетевых переменных

Обзор

Редактор NVL – **Declaration Editor** для редактирования Списков Сетевых Переменных (**Network Variables Lists)**. Редактор NVL работает, как Редактор объявлений и соответствует опциям, как офлайн, так и онлайн, установленным для текстового редактора. Объявление начинается с `VAR_GLOBAL` и заканчивается `END_VAR` ключевыми словами. Эти ключевые слова обеспечены автоматически. Введите между ними допустимые объявления *(см. стр.495)* глобальных переменных.

Редактор NVL

Раздел 18.2

Общая информация о сетевых переменных

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Введение в список сетевых переменных Network Variables List (NVL)	386
Конфигурирование обмена сетевых переменных	389
Правила списка сетевых переменных Network Variables List (NVL)	393
Рабочее состояние отправителя и получателя	395
Пример	396
Совместимость	401

Введение в список сетевых переменных (NVL)

Обзор

Функция Network Variables List (NVL) состоит из фиксированного списка переменных, которые могут быть отправлены или получены через коммуникационную сеть. Это разрешает обмен данными в сети через сетевые переменные, если поддерживается контроллером (целевой системой).

Список должен быть определен в контроллерах отправителя и получателя (и может поддерживаться в одном или в многочисленных проектах). Их значения передаются с помощью широковещательной передачи через дейтаграммы User Datagram Protocol (UDP). UDP - коммуникационный протокол Интернета без установления соединения, определенный IETF RFC 768. Этот протокол упрощает прямую передачу дейтаграмм в сетях Internet Protocol (IP). Сообщения UDP/IP не ожидают ответа и поэтому идеальны для приложений, в которых отброшенные пакеты не требуют повторной передачи (такой как потоковое видео и сети, которые требуют производительности реального времени).

Функциональность NVL - мощная функция SoMachine. Она позволяет Вам совместно использовать и контролировать данные между контроллерами и их приложениями. Однако нет никаких ограничений относительно цели данных, переданных между контроллерами, включая, но не ограничиваясь, попытками машины или процесса блокировки или даже изменений состояния контроллера.

Только Вы, разработчик приложений и/или программист, можете знать обо всех условиях и факторах, существующих во время работы машины или процесса, и поэтому только Вы можете определить надлежащие коммуникационные стратегии, взаимные блокировки и связанные средства безопасности, необходимые в Ваших целях при обмене данными между контроллерами, использующими эту функцию. Нужно соблюдать особое внимание, контроль этого типа коммуникационной функции и быть уверенным, что дизайн машины или процесса не представляет риска для безопасности людям или свойствам.

ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННАЯ РАБОТА МАШИНЫ ИЗ-ЗА НЕПРАВИЛЬНОЙ КОММУНИКАЦИИ МАШИНЫ

- Вы должны рассмотреть потенциальные виды отказа каналов управления и для определенных критических функций управления обеспечить средство достижения безопасного состояния в течение и после повреждения канала, включая системные перезапуски и перебои в питании.
- Раздельные или избыточные каналы управления должны быть обеспечены для критических функций управления.
- Вы должны рассмотреть последствия непредвиденных задержек передачи или отказы связи.
- Соблюдайте все инструкции техники безопасности и локальные правила техники безопасности.
- Каждая реализация оборудования, использующего эту функцию, должна быть индивидуально и полностью протестирована на правильное функционирование, прежде чем быть введённой в эксплуатацию.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

Вы можете использовать функциональные блоки Диагностики *(см. SoMachine, Network Variable Configuration, SE_NetVarUdp Library Guide)*, и Управления обработкой ошибок *(см. SoMachine, Network Variable Configuration, SE_NetVarUdp Library Guide)*, а также параметры сетевых свойств, чтобы контролировать исправность, состояние и целостность коммуникаций, использующих эту функцию. Эта функция была разработана для совместного использования данных и контроля и не может быть использована для критических функций управления.

Список сетевых переменных Network Variables List (NVL)

Сетевые переменные, которые участвуют в обмене, определены в следующих 2 типах списков:

- Списки глобальных переменных Global Variables Lists (GVL) в передающем контроллере (отправитель)
- Список глобальных переменных GlobalNetworkVariablesList (GNVL) в контроллере получения (получатель)

Соответствующие GVL и GNVL содержат одинаковые объявления переменных. Вы можете просмотреть их содержание в соответствующем редакторе, который открывается после двойного щелчка по узлу GVL или GNVL в области **Devices**.

GVL содержит сетевые переменные отправителя. В **Network properties** отправителя определены протокол и параметры передачи. Согласно этим параметрам настройки, значения переменных передаются в сеть. Они могут быть получены всеми контроллерами, у которых есть соответствующий GNVL.

ПРИМЕЧАНИЕ: Для обмена сетевыми переменными должны быть установлены соответствующие сетевые библиотеки. Это делается автоматически для стандартного типа сети UDP, как только установлены сетевые свойства для GVL.

Сетевые переменные передаются от GVL (отправителя) к одному или более GNVL (получатели). Для каждого контроллера Вы можете определить GVL, а также GNVL. Таким образом, каждый контроллер может действовать как отправитель, а также как получатель.

Отправитель GVL может быть обеспечен тем же или другим проектом. Таким образом, при создании GNVL отправитель GVL или может быть выбран из списка всех доступных GVL в сети, или он может быть считан из файла экспорта, который ранее был сгенерирован (например, при помощи диалогового окна **Link to File**) от GVL.

ПРИМЕЧАНИЕ: Файл экспорта необходим, если используемый отправитель GVL определен в другом проекте.

Рассмотрение NVL

Следующая таблица содержит список контроллеров, которые поддерживают функциональность списка сетевых переменных (NVL):

Имя функции	**M238**	**M241**	**M251**	**M258 LMC058**		**ATV IMC**
Список сетевых переменных	Нет	Да	Да	Да	Да	Да*
*ATV IMC поддерживает NVL только в свободных задачах.						

На рисунке показана сеть, состоящая из 1 отправителя Controller Sender A и рекомендуемых максимум 7 получателей Controller Reciver:

Controller Sender A: Отправитель со списком глобальных переменных (GVL) и контроллер получателя со списками глобальных сетевых переменных (GNVL)
Controller Receiver 1...7: Получатели (с GNVL) от A и контроллер отправителя (GVL) только для A

Конфигурирование обмена сетевыми переменными

Обзор

Для обмена сетевыми переменными между отправителем и получателем, один контроллер отправителя и один контроллер получателя должны быть доступны в дереве **Devicestree** SoMachine. Это контроллеры, которым присвоены сетевые свойства, описанные ниже.

Чтобы сконфигурировать список сетевых переменных, выполните следующее:

Шаг	Действие
1	Создайте контроллеры отправителя и получателя в **Devicestree**.
2	Создайте программу (POU) для контроллеров получателя и отправителя.
3	Добавьте задачу для контроллеров получателя и отправителя. **ПРИМЕЧАНИЕ:** Чтобы поддержать прозрачность действия, Вы должны установить приоритет задачи для специальной задачи NVL несколько больше чем 25 и отрегулировать коммуникации, чтобы избежать излишнего насыщения сети.
4	Определите список глобальных переменных (GVL) для отправителя.
5	Определите глобальный сетевой список переменных (GNVL) для получателя.

Пример с дополнительной информацией приведен в Приложении *(см. стр. 396)*.

Список глобальных переменных

Чтобы создать GVL для отправителя, определите следующие сетевые свойства в **GVL→Properties**

Диалоговое окно **Network properties**

Описание параметров:

Параметр	Значение по умолчанию	Описание
Network type	**UDP**	Доступен только стандартный тип сети **UDP**. Чтобы изменить **Broadcast Address** и **Port**, нажмите кнопку **Settings...**.
Task	**MAST**	Выберите задачу, которую Вы сконфигурировали ниже **Task Configuration** для выполнения кода NVL. Чтобы помочь поддерживать прозрачность производительности, мы рекомендуем сконфигурировать интервал времени цикла **Interval** ≥ 50 мс для этой задачи. **ПРИМЕЧАНИЕ:** Чтобы поддержать прозрачность производительности, Вы должны установить приоритет задачи для специальной NVL задачи несколько больше 25 и отрегулировать коммуникации, чтобы избежать излишнего насыщения сети.
List identifier	**1**	Введите уникальный номер для каждого GVL в сети. Он используется получателями для идентификации списка переменных *(см. стр. 394)*.
Pack variables	activated (активировано)	С этой активированной опцией переменные связаны в пакеты (дейтаграммы) для передачи. Если эта опция деактивирована, передается один пакет на переменную.
Transmit checksum	deactivated (деактивировано)	Активируйте эту опцию для добавления контрольной суммы к каждому пакету переменных во время передачи. Получатели при этом проверят контрольную сумму каждого пакета, который они получают, и отклонят те, контрольные суммы которых не совпадают. Уведомление будет выпущено с параметром `NetVarError_CHECKSUM` (см. *SoMachine, Network Variable Configuration, SE_NetVarUdp Library Guide*).
Acknowledgement	deactivated (деактивировано)	Активируйте эту опцию для запроса получателю отправлять сообщение подтверждения получения каждого пакета данных. Уведомление будет выпущено с параметром `NetVarError_ACKNOWLEDGE`*(см. SoMachine, Network Variable Configuration, SE_NetVarUdp Library Guide)*, если отправитель не получает это сообщение подтверждения от получателя, прежде чем он отправит следующий пакет данных.
Cyclic transmission ● **Interval**	activated (активировано)	Выберите эту опцию для циклической передачи данных в определенном **Интервале**. Этот **Interval** должен быть числом кратным времени цикла, которое Вы определили в задаче для выполнения кода NVL, чтобы достигнуть точного времени передачи сетевых переменных.
Transmit on change ● **Minimum gap**	deactivated (деактивировано) ● **T#20ms**	Выберите эту опцию для передачи переменных каждый раз, когда их значения изменились. **ПРИМЕЧАНИЕ**: После первой загрузки или использования команды **Reset Cold** или **Reset Warm** в режиме онлайн контроллеры получателей не обновляются и сохраняют свое последнее значение, тогда как значения контроллера отправителя становятся 0 (нуль). Параметр **Minimum gap** определяет минимальный отрезок времени, который должен пройти между передачами данных.
Transmit on event ● **Variable**	deactivated ●–	Выберите эту опцию для передачи переменных, пока указанная **Variable** равна TRUE. Переменная проверяется в каждом цикле задачи для выполнения кода NVL.

Описание кнопки **Settings...**

Параметр	Значение по умолчанию	Описание
Port	1202	Введите уникальный номер порта (≥ 1202) для каждого отправителя GVL.
Broadcast Address	255.255.255.255	Введите определенный широковещательный IP-адрес для Вашего приложения.

Список глобальных сетевых переменных Global Network Variables List (GNVL)

Список глобальных сетевых переменных может быть добавлен только в дереве **Devices**. Он определяет переменные, которые определены как сетевые переменные в другом контроллере в сети.

Таким образом, объект GNVL может быть добавлен к приложению, только если список глобальных переменных (GVL) с сетевыми свойствами (список сетевых переменных) был уже создан в одном из других сетевых контроллеров. Эти контроллеры могут быть в одном или различных проектах.

Чтобы создать GNVL, определите следующие параметры в диалоговом окне **AddObject→Global Network Variable List**:

Описание параметров

Параметр	Значение по умолчанию	Описание
Name (Имя)	**NVL**	Введите имя для GNVL.
Task (Задача)	Задача определена в узле **Task Configuration** этого **Application (Приложения)**	Выберите задачу из списка задач, которые примут фреймы от отправителя, которые доступны под узлом **Task Configuration** контроллера получателя.

Параметр	Значение по умолчанию	Описание
Sender (Отправитель)	1 из GVL, доступных в настоящее время в проекте	Выберите GVL отправителя из списка всех отправителей GVL с сетевыми свойствами, доступных в настоящее время в проекте. Выберите запись **Import from file** из списка, чтобы использовать GVL из другого проекта. Это активирует **Import from file:** параметр ниже.
Import from file: (Импорт файла:)	–	Этот параметр доступен только после того, как Вы выбрали опцию **Import from file** для параметра **Sender**. ... открывает стандартное окно Windows Explorer, которое позволяет Вам просматривать файл экспорта *.gvl, который Вы создали GVL в другом проекте. Для получения дополнительной информации обратитесь к представленному ниже параграфу *Как добавить GNVL из другого проекта*.

Как добавить GNVL в том же проекте

Когда Вы добавляете GNVL через диалоговое окно **AddObject**, все надлежащие GVL, которые найдены в текущем проекте для текущей сети, доступны для выбора в поле списка **Sender**. GVL из других проектов должны быть импортированы (см. ниже параграф *Как добавить GNVL из другого проекта*).

Из-за этого выбора, каждый GNVL в текущем контроллере (отправитель) соединен с 1 определенным GVL в другом контроллере (получатель).

Кроме того, Вы должны определить имя и задачу, которая ответственна за обработку сетевых переменных при добавлении GNVL.

Как добавить GNVL из другого проекта

Альтернативно прямому выбору отправителя GVL от другого контроллера, Вы можете также определить файл экспорта GVL, который Вы генерировали ранее от GVL при помощи свойств **Link to file**. Это позволяет Вам использовать GVL, который определен в другом проекте.

Чтобы достигнуть этого, выберите опцию **Import from file** для **Sender (Отправителя)**: параметр и определите путь в **Import from file:** параметр.

Вы можете изменить параметры настройки позже через диалоговое окно **Properties - GVL**.

Свойства GNVL

Если Вы дважды щелкнете по элементу **GNVL** в дереве **Devices**, его содержание будет отображено на правой стороне в редакторе. Но содержание GNVL не может быть отредактировано, потому что это - только ссылка на содержание соответствующего GVL. Точное имя и путь отправителя, который содержит соответствующий GVL, обозначены наверху области редактора вместе с типом используемого сетевого протокола. Если соответствующий GVL изменяется, содержание GNVL соответственно обновляется.

Правила списка сетевых переменных Network Variables List (NVL)

Правила об объеме данных

Из-за некоторых ограничений производительности соблюдайте следующие правила:

Номер	Правило
1	Передача данных от одного GVL (отправитель) к одному GNVL (получатель) не должна превышать 200 байт.
2	Обмен данными между несколькими GVL (отправители) одного контроллера и связанных с ними GNVL (получатели) не должен превышать 1000 байт переменных.

Правила о количестве дейтаграмм

Чтобы ограничить максимальное время цикла задач NVL, соблюдайте следующие рекомендации:

Номер	Правило	Описание
1	Ограничьте число полученных дейтаграмм за цикл до 20.	Когда предел превышен, остающиеся дейтаграммы рассматривают в следующем цикле. Формируется уведомление **Received overflow (**Полученное переполнение) в данных диагностики *(см. SoMachine, Network Variable Configuration, SE_NetVarUdp Library Guide)*. Одна дейтаграмма может содержать до 256 байт. Это означает, что Вы не должны превышать предел 5120 байт данных, полученных одним получателем.
2	Ограничьте число переданных дейтаграмм за цикл до 20.	Когда предел превышен, остающиеся дейтаграммы рассматривают в следующем цикле. Формируется уведомление **Transmit overflow** (переполнение передачи) в данных диагностики (см. *SoMachine, Network Variable Configuration, SE_NetVarUdp Library Guide*). Одна дейтаграмма может содержать до 256 байт. Это означает, что Вы не должны превышать предел 5120 байт данных, переданных на одном контроллере отправителя.

Если количество полученных / переданных дейтаграмм за цикл превышают предел в несколько раз, может произойти следующее:

- потеря UDP (протокол пользовательских дейтаграмм) дейтаграмм
- некогерентный или противоречивый обмен переменными

Адаптируйте следующие параметры согласно своим потребностям:

- время цикла контроллера отправителя
- время цикла контроллера получателя
- количество отправителей в сети

ЗАМЕЧАНИЕ

ПОТЕРЯ ДАННЫХ

Полностью протестируйте свое приложение на надлежащую передачу и прием дейтаграмм UDP до ввода Вашей системы в эксплуатацию.

Отказ следовать этим инструкциям может привести к повреждению оборудования.

Максимальное количество GVL (Senders - отправители)

Определите максимум 7 GVL на контроллер (отправитель), чтобы он мог поддерживать прозрачность производительности.

Правила о времени цикла задачи GVL (отправители) и GNVL (получатели)

Чтобы помочь избежать переполнения приема, Вы должны определить время цикла для задачи, которая управляет передачей GVL, по крайней мере, в два раза большее, чем время цикла задачи, которая управляет приемом GNVL.

Правила о защите идентификатора списка

Функция NVL включает проверку идентификатора списка:

Идентификатор списка помогает избежать того, чтобы GVL (отправитель) от двух раздельных контроллеров с одним и тем же идентификатором списка (см. диалоговое окно **GVL→Properties→List identifier:**) отправлял дейтаграммы в один и тот же список глобальных сетевых переменных (GNVL) любого контроллера. Если **List Identifier** не уникален, это может вызвать прерывание в обмене переменными.

ЗАМЕЧАНИЕ
ПОТЕРЯ КОММУНИКАЦИИ Гарантируйте, что идентификатор списка в сети используется только одним IP-адресом. **Отказ следовать этим инструкциям может привести к повреждению оборудования.**

Функция проверки идентификатора списка реализована в контроллере получателя.

Если GNVL (получатель) обнаруживает, что два различных IP-адреса используют один и тот же идентификатор списка, получатель немедленно прекращает получать дейтаграммы.

Кроме того, выпускается уведомление в функциональном блоке `NETVARGETDIAGINFO`. IP-адреса этих двух отправителей передаются в выходные параметры`dwDuplicateListIdIp1` и`dwDuplicateListIdIp2` этого функционального блока (см. *SoMachine, Network Variable Configuration, SE_NetVarUdp Library Guide*).

Обнаруженные ошибки NVL сбрасываются функциональным блоком `NETVARRESETERROR`, и коммуникация перезапускается.

Состояние работы отправителя и получателя

Состояние работы

Состояние работы...		Поведение сетевых переменных
Отправителя	Получателя	
RUN	RUN	Сетевые переменные обмениваются между отправителем и получателем.
STOP	RUN	Отправитель больше не отправляет переменные получателю. Сетевые переменные не обмениваются между отправителем и получателями.
RUN	STOP	Получатель не обрабатывает сетевые переменные от отправителя. Когда получатель возвращается в состояние RUN (Работа), сетевые переменные снова обрабатываются получателем.
STOP	STOP	Никакие переменные не обмениваются.

ПРИМЕЧАНИЕ: Обнаруживаются несколько коммуникационных ошибок инициализации (`NetVarError_INITCOMM`), когда Вы изменяете операционное состояние отправителя со STOP (остановка) на RUN (работа).

События в задаче, которая управляет NVL

Если следующие события имеют место в задаче, которая управляет NVL, ожидаемое для NVL поведение совпадает с тем, если контроллер был в состоянии STOP в таблице выше:

- исключение происходит в приложении, которое приостанавливает задачу
- контрольная точка поражена (is hit), или в задаче происходит обработка одиночного цикла.

Пример

Обзор

В следующем примере установлен простой сетевой обмен переменных. В контроллере отправителя создается список глобальных переменных (GVL). В контроллере получателя создается соответствующий список глобальных сетевых переменных (GNVL).

Выполните следующие приготовления в стандартном проекте, где контроллер отправителя **Dev_Sender** и контроллер получателя **Dev_Receiver** доступны в дереве **Devices**:

- Создайте POU (программа) **prog_sender** ниже узла **Application** контроллера **Dev_Sender**.
- Под узлом **TaskConfiguration** этого приложения добавьте задачу **Task_S**, которая вызывает **prog_sender**.
- Создайте POU (программа) **prog_rec** ниже узла **Application** контроллера **Dev_Receiver**.
- Под узлом **Task Configuration** этого приложения добавьте задачу **Task_R**, которая вызывает **prog_rec**.

ПРИМЕЧАНИЕ: Эти 2 контроллера должны быть сконфигурированы в той же подсети сети Ethernet.

Определение GVL для отправителя

Шаг 1: Определите список глобальных переменных в контроллере отправителя:

Шаг	Действие	Комментарий
1	В области **Devices** щелкните правой кнопкой по узлу **Application** контроллера **Dev_Sender** и выберите команды **Add Object→Global Variable List...**.	Отображается диалоговое окно **Add Global Variable List**.
2	Введите **Name** `GVL_Sender` и нажмите **Open**, чтобы создать новый список глобальных переменных.	Узел **GVL_Sender** появляется ниже узла **Application** в области **Devices**, и редактор открыт на правой стороне.

Шаг	Действие	Комментарий

3	В редакторе на правой стороне, введите следующие определения переменных:
	`VAR_GLOBAL`
	`iglobvar:INT;`
	`bglobvar:BOOL;`
	`strglobvar:STRING;`
	`END_VAR`
	
		–

Шаг 2: Определите сетевые свойства отправителя GVL:

Шаг	Действие	Комментарий
1	В области **Devices** щелкните правой кнопкой по узлу **GVL_Sender** и выберите команду **Properties...**.	Отображается диалоговое окно **Properties - GVL_Sender**.
2	Откройте вкладку **Network properties** и сконфигурируйте параметры как показано на изображении:	–
3	Нажмите **OK.**	Диалоговое окно закрыто, и GVL свойства сети установлены.

Определение GNVL для получателя

Шаг 1: Определите список глобальных сетевых переменных в контроллере получателя:

Шаг	Действие	Комментарий
1	Вобласти **Devices** щелкните правой кнопкой по узлу **Application** контроллера **Dev_Receiver** и выберите команды **Add Object** → **Global Network Variable List...**.	Отображается диалоговое окно **Add Global Network Variable List.**
2	Сконфигурируйте параметры как показано на изображении:	Этот список глобальных сетевых переменных - дубликат GVL, определенного для контроллера отправителя.
3	Нажмите **Open**.	Диалоговое окно закрыто, и **GNVL_Receiver** появляется ниже узла **Application** контроллера **Dev_Receiver**: Этот GNVL автоматически содержит те же объявления переменной, что и **GVL_Sender**.

Просмотр и / или изменение сетевых параметров настройки GNVL:

Шаг	Действие	Комментарий
1	В области **Devices** щелкните правой кнопкой по узлу **GNVL_Receiver** и выберите команду **Properties...**.	Отображается диалоговое окно **Properties - GVL_Sender**.
2	Откройте вкладку настройки сетевых параметров (**Network settings**).	–

Шаг 3: Тест сетевого обмена переменных в режиме онлайн:

Шаг	Действие	Комментарий
1	Под узлом **Application** контроллера **Dev_Sender** дважды щелкните по POU **prog_sender**.	Открывается редактор для **prog_sender** с правой стороны.

2	Введите следующий код для переменной `iglobvar`:
	prog_sender [Dev_Sender: Plc Logic: Application]
	`PROGRAM prog_sender`
	`VAR`
	`END_VAR`
	`iglobvar:=iglobvar+1;`
		–

3	Под узлом **Application** контроллера **Dev_Receiver** дважды щелкните по POU **prog_rec**.	Открывается редактор для **prog_rec** с правой стороны.

4	Введите следующий код для переменной `ivar_local`:
	prog_rec [Dev_Receiver: Plc Logic: Application]
	`PROGRAM prog_rec`
	`VAR`
	`ivar_local: INT;`
	`END_VAR`
	`ivar_local:=iglobvar;`
		–

5	Войдите в систему с приложениями отправителя и получателя в той же сети и запустите приложения.	Переменная `ivar_local` в получателе получает значения `iglobvar`, какое в настоящее время показано в отправителе.

Совместимость

Введение

Даже если контроллеры работают с приложениями различных версий системы программирования (например, V2.3 и V3.x), возможна коммуникация через сетевые переменные.

Однако форматы файлов различных файлов экспорта между версиями (*.exp по сравнению с *.gvl) лишают возможности просто импортировать и экспортировать эти файлы между проектами.

Если список глобальных сетевых переменных (GNVL) установлен в последней версии (например, V3.x), требуемая сетевая конфигурация параметров должна быть обеспечена отправителем последней версии (например, V3.x). Экспортный файл *.*exp*, создаваемый из отправителя более ранней версией (например, V2.3), не содержит эту информацию.

Решение для обмена сетевыми переменными между приложениями различных версий системы программирования предоставлено в следующих параграфах.

Обновление глобального списка сетевых переменных

Для обмена сетевыми переменными между приложениями различных версий системы программирования (например, V2.3 и V3.x), обновляют список глобальных сетевых переменных, выполняя следующие шаги:

Шаг	Действие	Комментарий
1	Воссоздайте в последней версии (V3.x) список сетевых переменных (NVL), который уже доступен в более ранней версии (V2.3).	Чтобы достигнуть этого, добавьте список глобальных переменных (GVL) с сетевыми свойствами, содержащий те же объявления переменных как в NVL более ранней версии (V2.3).
2	Экспортируйте новый GVL в *.*exp* файл при помощи вкладки **Link to File**.	**ПРИМЕЧАНИЕ**: Активируйте опцию **Exclude from build** на вкладке **Build**, чтобы сохранить GVL в проекте, не получая события перед компиляцией и неоднозначные имена. Деактивируйте опцию воссоздания *.*exp* файла снова в случае, если требуются любые модификации в GVL
3	Повторно импортируйте список.	Чтобы достигнуть этого, создайте новый список глобальных сетевых переменных (GNVL) при помощи ранее сгенерированного *.*exp* файла, чтобы создать соответственно сконфигурированный список получателя.

Эти шаги проиллюстрированы следующим примером.

Пример

В этом примере переменная `trans23`, которая определена в приложении V2.3, сделана доступной для более поздней версии (V3.x).

Определены следующие условия:

Условие	Описание

1	В более ранней версии (V2.3) системы программирования проект 23.pro содержит список глобальных переменных **GVL_23** со следующим объявлением:
	`VAR_GLOBAL`
	`trans23:INT;`
	`END_VAR`

2	Сетевые свойства GVL_23 сконфигурированы следующим образом:
	
	ПРИМЕЧАНИЕ: экспорт этого GVL_23 создает *.*exp* файл, который содержит только следующее объявление переменной:
	`VAR_GLOBAL`
	`trans23:INT;`
	`END_VAR`
	*.*exp* файл не содержит каких-либо параметров конфигурации.

Таблица показывает следующие шаги, которые выполняются для воссоздания GVL_23 в последней версии (V3.x):

Шаг	Действие	Комментарий
1	Добавьте объект GVL под названием **GVL_23** к приложению.	–
2	Установите сетевые свойства, как определено в проекте *23.pro*.	
3	На вкладке **Linkto File** сконфигурируйте целевой экспортный файл *23.gvl*.	
4	На вкладке **Build** установите опцию **Exclude from Build**.	Это установка позволяет Вам сохранять файл на диске для последующих модификаций.

Шаг	Действие	Комментарий
5	Скомпилируйте проект.	Файл *23.gvl* сгенерирован и содержит переменные и параметры конфигурации: <GVL> <Declarations><![CDATA[VAR_GLOBAL trans23: INT; END_VAR]]></Declarations> <NetvarSettings Protocol="UDP"> <ListIdentifier>1</ListIdentifier> <Pack>True</Pack> <Checksum>False</Checksum> <Acknowledge>False</Acknowledge> <CyclicTransmission>True</CyclicTransmission> <TransmissionOnChange>False</TransmissionOnChange> <TransmissionOnEvent>False</TransmissionOnEvent> <Interval>T#50ms</Interval> <MinGap>T#20ms</MinGap> <EventVariable> </EventVariable> <ProtocolSettings> <ProtocolSetting Name="Port" Value="1202" /> <ProtocolSetting Name="Broadcast Adr." Value="192.168.101.167" /> </ProtocolSettings> </NetvarSettings> </GVL>
6	Добавьте объект GNVL в проекте V3.x из файла экспорта *23.gvl* (с помощью **Import from file:** команда).	Это служит для того, чтобы считать переменную `trans23` из контроллера более ранней системы программирования (V.2.3). Если оба, проект от более ранней версии (V2.3), а также приложение от последней версии (V3.x), работают в сети, приложение от последней версии (V3.x) может считать переменную `trans23` из проекта *23.pro*.

Глава 19

Редактор задач

Содержание этой главы

Эта глава содержит следующие темы:

Тема	Страница
Информация о конфигурации задачи	406
Вкладка свойств	407
Вкладка монитора	408
Конфигурация определенной задачи	410
Обработка задачи в режиме онлайн	414

Информация о конфигурации задачи

Обзор

Конфигурация задачи определяет одну или несколько задач для управления обработкой прикладной программы. Таким образом, конфигурация задачи - существенный объект для приложения и должна быть доступной в **Applications Tree**.

Описание дерева конфигурации задачи

В самой верхней позиции дерева конфигурации задачи есть запись **Task Configuration** . Ниже есть определенные задачи, каждая представлена именем задачи. Вызовы POU определенных задач выведены на экран в дереве конфигурации задачи.

Вы можете редактировать дерево задачи (добавлять, копировать, вставлять или удалять задачи) надлежащими командами, применимыми для **Applications tree**. Например, для добавления новой задачи выберите узел **Task Configuration**, щелкните по кнопке зеленого плюса и выполните команду **Task...**. Альтернативно, Вы можете щелкнуть правой кнопкой по узлу **Task Configuration** и выполнить команду **Add Object → Task...**.

Сконфигурируйте определенные задачи в редакторе задач (*см. стр. 410*), который дополнительно обеспечивает вид мониторинга в режиме онлайн. Опции, доступные для конфигурации задачи, зависят от платформы контроллера.

Конфигурация задачи в дереве Приложений **(Applications tree)**

Задачи

Задача *(см. стр. 410)* используется, чтобы управлять обработкой программы IEC. Она определяется именем, приоритетом и типом, определяющим, какое условие инициирует запуск задачи. Вы можете определить это условие по времени (циклический, свободный) или по внутреннему или внешнему событию, которое инициирует задачу; например, нарастающий фронт глобальной переменной проекта или событие прерывания контроллера.

Для каждой задачи Вы можете определить ряд программ POUs, которые будут запущены задачей. Если задача будет выполняться в существующем цикле, эти программы будут обработаны в пределах 1 цикла.

Комбинация приоритета и условия определит, в каком хронологическом порядке (*см. стр. 414*) будут выполняться задачи.

Для каждой задачи Вы можете сконфигурировать управление временем (сторожевой таймер). Возможные параметры настройки зависят от определенной платформы контроллера.

Вкладка свойств

Обзор

Когда выбран узел **Task Configuration** (*см. стр. 406*), вкладка **Properties** будет открыта в виде редактора задачи.

Конфигурация задачи, вкладка **Properties**, пример

.

Информация о текущей конфигурации задачи будет отображена в соответствии с контроллером, например, максимально разрешённое число задач на тип задачи.

Вкладка монитора

Обзор

Если это поддерживается целевой системой, то функция мониторинга разрешена. Это - динамический анализ времени выполнения, количество вызовов и покрытия кода POU, которыми управляет задача. В режиме онлайн может контролироваться обработка задачи.

Онлайновое представление (вид) редактора задачи

Когда Вы выбираете главный узел в дереве **Task Configuration** помимо вкладки **Properties** (*см. стр. 407*), доступна вкладка **Monitor**. В режиме онлайн она показывает состояние и некоторую текущую статистику по циклам и время цикла в табличном представлении. Интервал обновления для значений совпадает с используемым для мониторинга значений контроллера.

Описание элементов

Когда главный узел в дереве **Task Configuration** выбран помимо диалогового окна **Properties** (*см. стр. 407*), на дальнейшей вкладке доступно диалоговое окно **Monitoring**. В режиме онлайн оно показывает состояние и некоторую текущую статистику по циклам, и времена цикла отображаются в табличном представлении. Интервал обновления значений совпадает с используемым для мониторинга значений контроллером.

Task	Status	IEC-Cycle Count	Cycle Count	Last Cycle Time (µs)	Average Cycle...	Max. Cycle...	Min...	Jitter (µs)	Min. Jitter (µs)	M
Main Task	Valid	6780	7071	9	12	124	7	1509	-15011	
t1	Valid	6780	7071	15	12	119	6	1497	-15021	

Конфигурация задачи, мониторинг

Для каждой задачи в строке отображается следующая информация:

Task (Задача)	Имя задачи, как определено в **Task configuration**.
State (состояние)	Возможные записи: • **Not created** (Не создана): не была запущена, начиная с последнего обновления; особенно используемые для задач по событиям • **Created** (Создана): задача известна в системе выполнения, но еще не установлена для работы • **Valid** (Допустима): задача находится в нормальном функционировании • **Exception** (Исключение): у задачи есть исключение
IEC-Cycle Count	Количество выполненных циклов после запуска приложения; 0, если функция не поддерживается целевой системой.
Cycle Count	Количество уже выполняемых циклов (в зависимости от целевой системы, оно может быть равным IEC Cycle Count или больше, если циклы считаются даже тогда, когда приложение не работает).
Last Cycle Time (µs)	Последнее измеренное время выполнения в мкс
Average Cycle Time (µs)	Среднее время выполнения всех циклов в мкс

Max. Cycle Time (µs)	Максимальное измеренное время выполнения из всех циклов в мкс
Min. Cycle Time (µs)	Минимальное измеренное время выполнения из всех циклов в мкс
Jitter (µs)	В последний раз измеренное дрожание (jitter)* в мкс
Min. Jitter (µs)	Минимальное измеренное дрожание* в мкс
Max. Jitter (µs)	Максимальное измеренное дрожание* в мкс
* дрожание: Время, которое проходит после запуска задачи, пока операционная система не показывает, что она работает.	

Чтобы сбросить значения в 0 для задачи, установите курсор на поле с именем задачи и выполните команду **Reset**, доступную в контекстном меню.

Конфигурация определенной задачи

Обзор

Когда Вы вставляете задачу в узел **Task Configuration** дерева **Applications tree**, редактор задачи для установки конфигурации задачи открывает вкладку **Configuration**.

Она открывается также, если Вы дважды щелкаете по доступной задаче (например, **MAST**), чтобы изменить конфигурацию задачи.

Вкладка **Configuration** для задачи

ПРИМЕЧАНИЕ: Вы можете изменить имя задачи, редактируя соответствующую запись в **Applications tree**.

Вставьте желаемые атрибуты.

Priority (Приоритет)	
Priority (0...31)	Число от 0... 31; 0 самый высокий приоритет, 31 является самым низким

Type (Тип)	
Список для выбора предлагает следующие типы задачи:	
Cyclic (Циклическая)	Задача будет обработана циклически согласно определению времени (время цикла задачи), заданному в поле **Interval** (см. ниже).
Freewheeling (Свободная)	Задача будет обрабатываться, как только программа запущена, и в конце одного выполнения она будет автоматически перезапущена в непрерывном цикле. Нет никакого определенного времени цикла.

Type (Тип)	
Status (Состояние)	Задача будет запущена, если переменная, определенная в поле **Event**, равна **TRUE**. **ПРИМЕЧАНИЕ:** Эта функция не доступна для всех поддерживаемых контроллеров. Для получения дополнительной информации обратитесь к Руководству по программированию Вашего контроллера.
Event (Событие)	Задача будет запущена, как только переменная, определенная в поле **Event**, будет иметь нарастающий фронт.
External event (Внешнее событие)	Задача будет запущена, как только произойдет системное событие, которое определено в поле **Event**. Оно зависит от целевой платформы, чьи события будут поддерживаться и предлагаться в списке выбора. (Не путать с системными событиями.)

Различие между состоянием и событием

Когда указанное событие, являясь ИСТИНОЙ (TRUE), выполняет условие запуска задачи, управляемой по состоянию, тогда как задача, управляемая по событию, требует изменения события от FALSE до TRUE. Если событие изменяется слишком быстро от TRUE до FALSE и назад к TRUE, то это событие может быть оставлено необнаруженным, и таким образом задача **Event** не будет запущена.

Следующий пример иллюстрирует получающееся поведение задачи в реакции на событие (зеленая линия):

При выборке точек 1...4 задачи различных типов показывают различную реакцию:

Поведение в точке:	1	2	3	4
Status (Состояние)	Не запущена	Запущена	Запущена	Запущена
Event (Событие)	Не запущена	Запущена	Не запускается, потому что событие изменилось слишком быстро от TRUE до FALSE и назад к TRUE	Не запущена

Обязательные записи в зависимости от выбора задачи

Запись	Описание
Interval task cycle time **Время цикла интервала задачи**	Обязательная для типа **Cyclic** (циклического). Время (в миллисекундах [ms]), после которого задача должна быть перезапущена. **ПРИМЕЧАНИЕ**: Рассмотрим используемую в настоящее время магистральную систему при установке времени цикла задачи. Например, на шине CAN Вы можете установить **Bus cycle task** (задачу по циклу шины**)** на вкладке **CANopen I/O Mapping**. Это должно соответствовать установленной в настоящее время скорости передачи и количеству фреймов, используемых на шине. Далее времена, установленные для подтверждения работоспособности, защиты узла и синхронизации, всегда должны быть кратны времени цикла задачи. Иначе фреймы CAN могут потеряться. Для получения дополнительной информации обращайтесь к части *Device Editor* онлайн-справки SoMachine.
Event **Событие**	Обязательная для типа **Event** (по событию) или инициированного **External event** (внешним событием**)**. Глобальная логическая переменная, которая инициирует запуск задачи, как только обнаружен нарастающий фронт. Используйте кнопку**...** или **Input Assistant**, чтобы получить список всех доступных глобальных переменных события. **ПРИМЕЧАНИЕ**: Если событие, которое управляет задачей, происходит от входа, должна быть, по крайней мере, одна задача, которая не управляется событиями. Иначе Вх/Вых никогда не будут обновляться, и задача никогда не будет запускаться.

Параметры настройки сторожевого таймера

Для каждой задачи Вы можете сконфигурировать контроль по времени (сторожевой таймер).

Настройки по умолчанию сторожевого таймера зависят от Вашего контроллера.

Когда активирована опция **Enable** (поставлена галочка), сторожевой таймер включен. Когда сторожевой таймер задачи включен, обнаруживается ошибка **Exception** (исключения), если время выполнения задачи превышает определенное ограничение по времени задачи (**Time**) относительно определенной **Sensitivity**. Если опция **Update IO while in stop** включена в диалоговом окне параметров настройки контроллера, выходы будут установлены в предопределенные по умолчанию значения в зависимости от конкретной платформы контроллера.

Time (например, t#200ms)	Определяет допустимое максимальное время выполнения для задачи. Когда задача займет больше времени, чем допустимое, контроллер сообщит о сторожевом исключении задачи.
Sensitivity	Определяет количество сторожевых исключений задачи, которые должны произойти, прежде чем контроллер обнаружит ошибку приложения.

ПРИМЕЧАНИЕ: сторожевая функция недоступна в режиме симуляции.

Для получения дополнительной информации обратитесь к главе *System and Task Watchdog* Руководства по программированию Вашего контроллера.

Элементы организации программы (POU)

Программные единицы (POU), которые управляются задачей, перечисляются здесь с именем POU и опцией комментария **Comment**. Выше таблицы имеются команды для редактирования:

- С целью определения новой POU откройте диалоговое окно **Input Assistant** командой **Add Call**. Выберите одну из программ, доступных в проекте. Вы также можете добавлять POU в список методом перетаскивания из **Applications tree**.
- Для того, чтобы заменить вызов одной программы на другую, выберите запись в таблице, откройте диалоговое окно **Input Assistant** командой **Change Call...** и выберите другую программу.
- Для того, чтобы удалить вызов, выберите его в таблице и используйте команду **RemoveCall**.
- Команда **OpenPOU** открывает выбранную в данный момент программу в соответствующем редакторе.

Последовательность перечисленных вызовов POU сверху вниз определяет последовательность выполнения в режиме онлайн. Вы можете перемещать выбранную запись в пределах списка командами **Moveup** (переместить вверх) и **Movedown** (переместить вниз).

Строковые функции

Строковые функции стандартной библиотеки (обращайтесь к секции *CoDeSys Libraries / Standard Library* онлайн справки SoMachine) не запрограммированы для повторного обращения.

ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННАЯ РАБОТА ОБОРУДОВАНИЯ

Не используйте одну и ту же строковую функцию в нескольких задачах, потому что это может вызвать ошибки программы, из-за перезаписи.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

Чтобы помочь в преодолении этого ограничения, Вы можете захотеть использовать *Synchronization of I EC Tasks* из CAA_Types.library (обратитесь к разделу *CoDeSys Libraries / CAA_Types.library* онлайн-справки SoMachine).

Обработка задачи в режиме онлайн

Какая задача обрабатывается?

Для выполнения задач, определенных в **Task Configuration**, применяются следующие правила:

- Выполняется та задача, чье условие соблюдено. Это означает, если истекло требуемое время, или после того, как её переменная условия (события) показывает нарастающий фронт.
- Если допустимое требование будет у нескольких задач, то будет выполняться задача с самым высоким приоритетом (**Priority)**.
- Если у нескольких задач будут допустимые условия и эквивалентные приоритеты, то сначала будет выполняться задача с самым долгим временем ожидания.
- Обработка вызовов POU (типа программа) будет сделана согласно их порядку (сверху вниз) в **Task Editor**. Если вызывают POU, которая доступна с тем же именем как в **Applications tree**, присвоенном приложению, так и в библиотеке или в глобальном для проекта узле **Global** дерева **Applications tree**, то будет выполняться та, которая непосредственно объявлена ниже в **Applications tree**.

Глава 20

Редактор списка наблюдения

Содержание этой главы

Эта глава содержит следующие темы:

Тема	Страница
Вид наблюдения / Редактор списка наблюдения	416
Создание списка наблюдения	417
Просмотр списка в режиме онлайн	419

Вид наблюдения / Редактор списка наблюдения

Обзор

Список наблюдения - определяемый пользователем набор переменных проекта. Они отображаются на экране в виде просмотра для мониторинга (*см. стр. 419*) значений в таблице. Кроме того, в этом виде возможны запись и форсированная установка переменных.

Откройте вид наблюдения через подменю команды **Watch** (по умолчанию в меню **View**). Он предоставляется редактору для создания списков наблюдения (*см. стр. 417*).

По умолчанию Вы можете установить 4 отдельных списка наблюдения **Watch1**, **Watch2**, **Watch3**, **Watch4**. Вид **Watch all Forces** (просмотр всех форсированных переменных) в режиме онлайн автоматически заполняется *(см. стр. 420)* всеми принудительными в настоящее время значениями активного приложения.

Создание списка наблюдения

Обзор

Чтобы установить список наблюдения **Watch <n>** в виде **Watch**, щелкните в поле столбца **Expression** и нажмите клавишу пробела (SPACE), чтобы отредактировать столбец **Expression**. Введите полный путь для желаемого выражения просмотра. **Input Assistant** доступен через кнопку **...**.

Синтаксис выражения Expression для наблюдения

<имя устройства>. <имя приложения>. <имя объекта>. <имя переменной>

Пример

Пример:

`Dev1.App1.PLC_PRG.ivar`

Тип переменной обозначен значком перед именем переменной:

Значок	Переменная
	Вход
	Выход
	Обычная

После ввода переменной в столбец Expression, соответствующий тип данных добавляется автоматически в столбец **Type**. Если к объявлению переменной был добавлен комментарий, он добавляется в столбце **Comment**.

Столбец **Value** отображает текущее значение выражения в режиме онлайн (*см. стр. 419*).

Чтобы подготовить значение для переменной, щелкните в поле присвоения столбца **Prepared value** и непосредственно введите требуемое значение.

В случае логической переменной обработка еще проще: Вы можете переключать булевы значения подготовки, используя клавиши RETURN или SPACE согласно следующему порядку:

Если значение TRUE, шаги подготовки - FALSE -> TRUE -> нет записи; иначе, если значение - FALSE, шаги подготовки - TRUE -> FALSE -> нет записи.

Сделайте то же для желаемых последующих выражений/переменных в последующих строках. Посмотрите пример в следующем изображении, которое показывает вид наблюдения в режиме офлайн: Он содержит выражения объектов `PLC_PRG` и `Prog_St`.

Следует иметь в виду, что в случае структурированной переменной, как с экземпляром функционального блока, определенные компоненты экземпляра автоматически добавляются, когда Вы вводите имя экземпляра (см. в примере: `Dev1.App1.PLC_PRG.fbinst`). Щелкните по знаку плюса/минуса для отображения или скрытия их в складке.

Пример, вид наблюдения в режиме офлайн

Watch 1

Expression	Comment	Type	Value	Prepared va
Device.Appli...	counter	INT	<Not logged in>	
Device.Appli...	instance of FB1	FB1		
fbin		INT	<Not logged in>	
fbout		INT	<Not logged in>	
fbvar		INT	<Not logged in>	

В режиме онлайн (*см. стр. 419*) Вы можете использовать список для мониторинга значений переменных.

ПРИМЕЧАНИЕ: В режиме онлайн Вы можете добавить выражения к списку наблюдения при помощи команды **AddWatch**.

Список наблюдения в режиме онлайн

Мониторинг

Список наблюдения (*см. стр. 416*) (**Watch <n>**) в режиме онлайн показывает текущее значение переменной в столбце **Value**. Это значение, которое переменная имеет между двумя соседними циклами задачи.

Также возможно отображение назначенного прямого IEC адреса и/или комментария. Компоненты вида соответствуют тем же компонентам из онлайнового вида редактора объявлений (*см. стр. 376)*.

См. главу *Создание списка наблюдения* (*см. стр. 417*) с описанием того, как установить такой список наблюдения и как использовать складки в случае структурированных переменных.

Вид **Watch** в режиме онлайн

Watch 1

Expression	Type	Value	Prepared value	Address	Comment
myDev.Application.PLC_PRG.myStruct	TestStruct				Check address defined in "TestStruct"
nTest1	INT	11876		%MW0	
nTest2	INT	0		%MW2	
myDev.Application.PLC_PRG.bVar	BOOL	TRUE		%QX0.3	
myDev.Application.PLC_PRG.fbinst	FB1				instance of function block FB1
fbin	INT	0			
fbout	INT	0			
fbvar	INT	0			

Messages – Totally 0 error(s), 0 warning(s), 0 message(s) | Watch 1

ПРИМЕЧАНИЕ: В режиме онлайн Вы можете добавить выражения к списку наблюдения при помощи команды Add Watch.

Запись и форсированная установка значений

В столбце **Prepared value** Вы можете ввести требуемое значение, которое будет записано или принудительно установлено в соответствующее выражение на контроллере командами **Write values** или **Force values**. Обратитесь к описаниям команд **Write** и **Force**, применимых также в других видах мониторинга (например, редакторе объявлений).

Наблюдение за всеми форсированными переменными

Это – специальный вид списка наблюдений, который в режиме онлайн автоматически заполнен текущими значениями всех в настоящее время форсированных значений активного приложения. Каждое **Expression**, **Type**, **Value** и **Prepared value** будут показаны, как в онлайновом виде списка **Watch <n>.**

Вы можете отменить форсированные значения одной из следующих команд, доступных через кнопку **Unforce...**:

- **Unforce all selected Expressions, without modifying the value.** Отмена форсирования всех выбранных выражений без изменения значений.
- **Unforce all selected Expressions and restore the variable to the value it had before forcing it.** Отмена форсирования всех выбранных выражений и возврат переменной значения, которое она имела до форсирования.

Диалоговое окно наблюдения **Watch all Forces**

Глава 21

Инструменты в логических редакторах

Содержание этой главы

Эта глава содержит следующие темы:

Тема	Страница
Средство поиска функции и функционального блока	422
Ассистент ввода	425

Средство поиска функции и функционального блока

Обзор

SoMachine обеспечивает средство поиска FFB (функция и функциональный блок), которое помогает Вам в нахождении определенной функции или функционального блока, даже если Вы не знаете его точное имя.

Вы можете использовать средство поиска функции и функционального блока в следующих языках программирования, которые позволяют вставлять функциональные блоки:

- CFC
- LD
- IL
- FBD
- ST

Как найти функцию или функциональный блок с помощью средства поиска FFB

Если Вы собираетесь создать код программы в SoMachine Logic Builder, перейдите к месту, где Вы хотите вставить функциональный блок и откройте средство поиска FFB следующим образом:

- выберите в меню **Edit** → **FFBFinder**

или

- щелкните правой кнопкой по соответствующему месту в редакторе и выберите команду **FFB Finder...** из контекстного меню

Открывается Диалоговое окно **FFB Finder**:

Диалоговое окно **FFB Finder** содержит следующие элементы для нахождения функции или функционального блока:

Элемент	Описание
Find what **(Что найти)**	В текстовое поле **Find what** введите имя функции или функционального блока, который Вы хотите вставить в Ваш код программы. Как подстановочные знаки Вы можете использовать вопросительный знак (?), который заменяет точно один символ или звездочку (*), который может заменить несколько символов или никаких символов вообще.
Company **(Компания)**	Если Вы знаете компанию, которая создала библиотеку, включающую функциональный блок, который Вы ищете, Вы можете выбрать следующие компании из списка **Company**: ● **3S - Smart Software SolutionsGmbH** ● **CAA Technical Workgroup** ● **Schneider Electric** ● **System** Этот параметр по умолчанию установлен на **All companies** (все компании).

Элемент	Описание
Match case **(С учётом регистра)**	Установите флажок **Match case**, чтобы выполнить поиск с учетом регистра. По умолчанию этот флажок не установлен.
Include comments in search **(Включая комментарии в поиск)**	Установите флажок **Include comments in search**, чтобы искать введенную последовательность не только в именах функций и функциональных блоков, но также и в комментариях, которые сохранены с ними. По умолчанию этот флажок не установлен.
Search in project Libraries only **(Поиск только в библиотеках проекта)**	Установите флажок **Search in project libraries only**, чтобы ограничить поиск теми библиотеками, которыми пользуются в текущем приложении. По умолчанию этот флажок не установлен, и поиск включает все библиотеки, которые установлены в SoMachine PC.
Find **(Найти)**	Нажмите кнопку **Find** или нажмите клавишу ENTER, чтобы начать поиск функции или функционального блока.

Результаты, возвращаемые средством поиска FFB

Любая функция или функциональный блок, которые соответствуют введенным критериям поиска, будут перечислены в списке **Results** со следующей информацией:

- **Name (Имя)** функции или функционального блока
- **Library (Библиотека)**, в которой сохранена функция или функциональный блок
- **Version (Версия)** библиотеки
- **Company (Компания)**, которая создала библиотеку
- Комментарий, при наличии, будет выведен на экран в столбце с правой стороны.
- Столбец **Loaded** с левой стороны указывает, сохранена ли библиотека, функция или функциональный блок в уже используемом текущем проекте.

Чтобы отобразить дополнительную информацию об одной из функций или функциональных блоков, выберите его из списка. В поле ниже будут отображены изображение функции / функциональный блок с его вводами и выводами, а также описание или дальнейшая информация, при их наличии.

Вставка Функции / Функционального блока в Код программы

Чтобы вставить функцию / функциональный блок, который был найден посредством поиска FFB, в Ваш код программы, выберите его в списке **Results**, и

- или двойным щелчком выберите запись в списке **Results**,
- или нажмите кнопку **OK.**

Выбранная функция / функциональный блок будет вставлен в месте, где расположен курсор в Вашем коде программы, и соответствующая библиотека будет загружена автоматически.

Повторите эту работу каждый раз, когда Вам нужна помощь в нахождении определенной функции / функционального блока.

Ассистент ввода

Обзор

Диалоговое окно **Input Assistant** (Ассистент ввода) и соответствующая команда **Input Assistant** (по умолчанию в меню **Edit** → **Smart Coding**) доступны только тогда, если курсор установлен в окно текстового редактора. Диалоговое окно предлагает доступные элементы проекта для того, чтобы быть вставленными в текущую позицию курсора.

Диалоговое окно **Input Assistant**

Описание элементов

Диалоговое окно **Input Assistant** обеспечивает следующие элементы:

Элемент	Описание
Categories (Категории)	В этой области элементы проекта отсортированы по **Категориям**.
Filter (Фильтр)	Вы можете установить **Фильтр** для категории **Variables** (Переменные). Для отображения определенного типа переменной выберите запись из списка, такую как **Local variables (Локальные переменные)**, **Global variables (Глобальные переменные)**, **Constants (Константы)**.
Область **Items**	
Name (Имя), **Type (Тип)**, **Address (Адрес)**, **Origin (Источник)**	Область **Items** показывает доступные элементы и - в зависимости от категории - также **Type**, **Address** и **Origin** их данных, для категории, выбранной в области **Categories**. **Источник** показан для переменных ввода-вывода (путь в **Devices Tree**) и определенных библиотекой переменных (имя библиотеки и категория). Вы можете сортировать элементы по **Name**, **Type**, **Address**, **Origin** в возрастающем или убывающем алфавитном порядке. Чтобы произвести это, щелкните в соответствующем заголовке столбца (символ стрелки вверх или стрелки вниз). Чтобы скрыть или вывести на экран столбцы **Type**, **Address** или **Origin**, щелкают правой кнопкой мыши по заголовку соответствующего столбца.
Structured view (структурированный вид)	Если выбран вид опции **Structured**, элементы проекта выведены на экран в дереве структуры, добавленном значками. Если опция не выбрана, элементы проекта расположены плоско. Каждый элемент проекта выведен на экран с POU, которому он принадлежит (пример: **GVL1.gvar1**).

ПРИМЕЧАНИЕ: Если есть объекты с тем же именем, доступным в узле **Global Applications Tree**, а также ниже приложения (**Applications Tree**), в **Input Assistant** предлагается только 1 запись, потому что использование объекта определено обычными приоритетами вызова (сначала присвоенный приложению объект, потом глобальный).

Элемент	Описание
Show documentation (показать документацию)	Если выбрана опция **Show documentation**, диалоговое окно **Input Assistant** расширено полем **Documentation**. Если выбранный элемент - переменная, и ей присвоен адрес, или в его объявлении был добавлен комментарий, они отображаются здесь.
Insert with arguments (вставка с аргументами)	Если выбрана эта опция, элементы, которые включают аргументы, например, функции, вставлены с этими аргументами. Пример: Если функциональный блок FB1, который содержит входную переменную `fb1_in` и выходную переменную `fb1_out`, будет вставлен с аргументами, в редакторе будет записано следующее: `fb1(fb1_in:= , fb1_out=> )`
Insert with namespace prefix	Если выбрана эта опция, элемент вставлен со снабженным префиксом пробела именем. В настоящее время эта опция доступна только для глобальных переменных

Часть VI

Инструменты

Содержание этой части

Эта часть содержит следующие главы:

Глава	Название главы	Страница
22	Регистрация данных	429
23	Менеджер рецептов	431
24	Редактор трассировки	449
25	Редактор конфигурации символов	473
26	Обмен данными между контроллером SoMachine и HMI	481

Глава 22

Регистрация данных

Введение в регистрацию данных

Обзор

Вы можете контролировать и анализировать данные приложения, исследовав файл журнала данных (.log).

Рисунок показывают приложение, которое содержит два функциональных блока, `LogRecord` и `Dump`. Функциональный блок `LogRecord` записывает данные в буфер, который опорожняется в файл журнала данных (.log), размещённый в памяти контроллера. Буфер сбрасывается автоматически, когда заполнен на 80% или это может быть вызвано функцией `Dump`. Как стандартый клиент FTP, PC может получить доступ к этому файлу журнала данных, когда контроллер действует как FTP-сервер. Возможна также загрузка файла с SoMachine или веб-сервера контроллера.

ПРИМЕЧАНИЕ: Только контроллеры с функцией управления файлами могут поддерживать регистрацию данных. Обратитесь к своему руководству по программированию контроллера, чтобы видеть, поддерживает ли он управление файлами. Само программное обеспечение не оценивает Ваш контроллер для совместимости с действиями регистрации данных.

Пример файла журнала регистрации данных (.log)

```
Entries in File: 8; Last Entry: 8;
18/06/2009;14:12:33;cycle: 1182;
18/06/2009;14:12:35;cycle: 1292;
18/06/2009;14:12:38;cycle: 1450;
18/06/2009;14:12:40;cycle: 1514;
18/06/2009;14:12:41;cycle: 1585;
18/06/2009;14:12:43;cycle: 1656;
18/06/2009;14:14:20;cycle: 6346;
18/06/2009;14:14:26;cycle: 6636;
```

Процедура реализации

До начала написания Вашей программы сначала объявите и сконфигурируйте файлы журнала данных в своем приложении.

Глава 23

Менеджер рецептов

Содержание этой главы

Эта глава содержит следующие темы:

Тема	Страница
Менеджер рецептов	432
Определение рецепта	435
Команды `RecipeMan`	439

Менеджер рецептов

Обзор

Функции **RecipeManager** доступны только если выбраны в используемом в настоящее время наборе функций (**Options** → **Features** → **Predefined feature sets.**)

Менеджер рецептов обеспечивает функции для обработки определяемых пользователем списков переменных проекта, названных определениями рецептов, и определяет наборы значений для этих переменных в определении рецепта, называемых рецептами.

Вы можете использовать рецепты для установки и наблюдения параметров управления в контроллере. Они могут также быть загружены из файлов и сохранены в них. Эти взаимодействия возможны при помощи элементов визуализации, которые Вы соответственно должны сконфигурировать (конфигурация входа выполнения команды). Вы можете также использовать определенные команды рецепта в приложении (*см. стр. 439*).

Когда Вы выбрали рецепт, проверьте, подходит ли рецепт для процесса, которым будут управлять.

ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННАЯ РАБОТА ОБОРУДОВАНИЯ

- Проведите анализ безопасности для приложения и установленного оборудования.
- Проверьте, что рецепт подходит для процесса и оборудования или функции в установке.
- Установите надлежащие параметры, особенно для пределов и других связанных с безопасностью элементов.
- Проверьте, что все датчики и приводы совместимы с выбранным рецептом.
- Полностью протестируйте все функции во время проверки и ввода в действие.
- Обеспечьте независимые пути для критических функций управления (чрезвычайная остановка, условия превышения ограничений и т.д.) согласно анализу безопасности и применимым кодам, и инструкциям.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

По умолчанию менеджер рецептов загружается в контроллер во время загрузки. Он обрабатывает запись и чтение рецептов, когда приложение работает на контроллере. Однако нет необходимости загружать менеджера рецептов в контроллер, чтобы использовать рецепты только для обмена параметрами во время запуска системы (это - когда SoMachine все еще соединен с контроллером). Вы можете деактивировать его загрузку, используя для этой цели управление опцией **Recipe management in the plc**. Запись и чтение значений рецепта тогда будут обработаны стандартными онлайновыми командами и службами. Если управление рецептом должно работать на контроллере, потому что это необходимо прикладной программе в течение времени выполнения, то функциональный блок `RecipeCommands` ответственен за обработку команд рецепта.

Для описания поведения рецептов в различных режимах онлайн обратитесь к главе *Описание рецептов* (*см. стр.435*).

Если менеджер рецептов расположен на другом контроллере, не находящемся под воздействием приложения, затрагиваемого рецептами, сервер данных будет использоваться для чтения/записи переменных, содержащихся в рецептах.
Чтение и запись переменных делаются синхронно. Вызывая `g_RecipeManager`. после чтения/записи `LastError`, Вы можете проверить, была ли передача выполнена успешно (в этом случае `g_RecipeManager.LastError=0`).

Объекты управления рецептами в Tools Tree

Чтобы добавить объект менеджера рецептов к **Tool stree**, выберите узел **Application**, щелкните по кнопке зеленого плюса и выполните команду **Add other objects...** → **Recipe Manager...**. Подтвердите диалоговое окно **Add Recipe Manager** щелчком по **Add,** и узел **Recipe Manager** вставляется ниже узла **Application** (приложения).

Вы можете добавить один или несколько объектов **Recipe Definition** в узел **Recipe Manager**. Чтобы произвести это, щелкните по кнопке зеленого плюса узла **Recipe Manager** и выполните команду **Recipe Definition...**. Введите **Name (Имя)** в диалоговом окне **Add Recipe Definition** и нажмите **Add (Добавить)**. Дважды щелкните по узлу, чтобы просмотреть и отредактировать определения рецепта, включая определенные рецепты в отдельном окне редактора. Для описания поведения рецептов в различных онлайн режимах обратитесь к главе *Определение рецептов* *(см. стр.435)*.

Редактор Менеджера рецептов, вкладка сохранения

По умолчанию рецепты будут сохранены автоматически в файлах, согласно параметрам настройки на вкладке **Storage** редактора **Recipe Manager**:

Параметр	Описание
Storage Type	Выберите **Textual (Текстовый)** или **Binary** (**Двоичный**) тип сохранения
File Path	Определите месторасположение, где рецепт будет сохранен.
File Extension	Определите расширение для файла рецепта.

ПРИМЕЧАНИЕ: Файл сохранения может также быть определен вводом на элементе визуализации (входная конфигурация - выполняется команда - сохранение/загрузка рецепта из файла). Однако при определении имени такого файла в конфигурации визуализации, не перезаписывайте **.txt recipe* файл, определенный здесь в менеджере рецептов.

Параметр	Описание
Separator (разделитель)	В случае текстового сохранения столбцы, выбранные для сохранения, будут разделены разделителями. Выберите 1 из 6 предложенных опций.
Available Columns (Доступные столбцы)	Все столбцы определения рецепта представлены соответствующим заголовком.
Selected Columns (Выбранные столбцы)	Выбраны столбцы определения рецепта, т.е. столбцы, которые будут сохранены. По крайней мере, столбец, содержащий текущее значение (**Current Value)**, включен в эту часть. Он не может быть невыбранным.
Кнопки стрелок	Другие столбцы могут быть смещены направо или налево выбором соответствующей записи и нажатием кнопки стрелки. Вы можете также сместить все записи от одной стороны до другой сразу при помощи кнопок двойных стрелок.
Кнопки **Up** и **Down**	Нажмите эти кнопки, чтобы скорректировать порядок выбранных столбцов, который представляет порядок столбцов в файле хранения. Для каждого рецепта файл <recipename>.<recipedefinition>.<fileextension>; будет создаваться в указанной папке. Этот файл будет перезагружен менеджеру рецептов при каждом перезапуске приложения. Для конфигурации обновления файлов сохранения рецепта обратитесь к описанию вкладки "**General**" (*см. стр. 434*)
Save as default	Щелкните по кнопке **Save as default**, чтобы использовать настройки, установленные в этом диалоговом окне, поскольку настройки по умолчанию для каждоЙ дальнейшей вставки менеджера рецептов.
Save changes to recipes automatically	Выберите эту опцию для обновления файлов сохранения сразу после любой модификации рецепта в реальном времени.

Редактор Менеджера рецептов, вкладка General

Параметр	Описание
Recipe management in the plc	Если менеджер по рецепту не необходим на контроллере, потому что никакие рецепты не должны быть обработаны в течение времени выполнения приложения, Вы можете деактивировать эту опцию. Это отменит загрузку менеджера. Автоматическое обновление файла рецепта возможно только после того, как загрузка была выполнена. Чтобы загрузить управление рецептами в контроллер, выберите эту опцию.

Определение рецепта

Обзор

Менеджер рецептов (*см. стр. 432*) обрабатывает одно или несколько определений рецепта. Определение рецепта содержит список переменных и один или несколько рецептов (наборов значений) для этих переменных. При помощи различных рецептов Вы можете присвоить другой набор значений ряду переменных на контроллере за один раз. Нет никакого ограничения количества определений рецептов, рецептов и переменных в рецепте.

Определение рецепта

Вы можете добавить один или несколько объектов **Recipe Definition** в узел **RecipeManager** в дереве **Toolstree**. Чтобы произвести это, щелкните по кнопке зеленого плюса узла **Recipe Manager** и выполните команду **RecipeDefinition...**.

Дважды щелкните по узлу, чтобы просмотреть и отредактировать определения рецепта, включая определенные рецепты в отдельном виде редактора.

Вид редактора определения рецепта

1 имя определения рецепта
2 имена рецепта

Окно редактора будет названо именем определения рецепта.

Параметр	Описание
Variable (Переменная)	В таблице Вы можете ввести несколько переменных проекта, для которых Вы хотите определить 1 или несколько рецептов. С этой целью Вы можете использовать команду **Insert Variable**, когда курсор находится в любом поле любой строки. Альтернативно, Вы можете дважды щелкнуть по полю **Variable**, или Вы можете выбрать его и нажать клавишу "Пробел", чтобы войти в режим редактора. Введите допустимое имя переменной проекта, например `plc_prg.ivar`. Щелкните кнопку **...**, чтобы открыть ассистент ввода.
Type (Тип)	Поле **Type** заполняется автоматически. Дополнительно, Вы можете определить символьное имя **Name**.
Name (Имя)	Вы можете определить символьное **имя**.
Comment	Введите дополнительную информацию, такую как единицы значения, зарегистрированного в переменной.

Параметр	Описание
Minimal Value и **Maximal Value**	Вы можете дополнительно определить те значения, которые должны быть допустимыми для того, чтобы быть записанными в этой переменной.
Current Value	Это текущее значение контролируется в режиме онлайн.
Save changes to recipes automatically	Это - наилучшая практика активации этой опции, потому что она влияет на обычное поведение управления рецептом: файлы хранения будут сразу обновлены при любой модификации рецепта во время работы. Полагайте, что опция может быть эффективной только пока менеджер рецептов доступен в контроллере.

Вы можете удалить переменную (строку) из таблицы, нажав клавишу DEL, когда выбрана одна из её ячеек. Вы можете выбрать несколько строк, удерживая клавишу CTRL нажатой при выборе ячеек. Вы можете скопировать выбранные строки командами копии и вставки. Вставка вставляет скопированные строки выше выбранной в настоящее время. При этом значения рецепта будут вставлены в соответствующий столбец рецепта, если доступны.

Чтобы добавить рецепт к определению рецепта, выполните команду **Adda new recipe** (см. *SoMachine, Menu Commands, Online Help*), когда фокус находится в поле редактора. Для каждого рецепта будет создан собственный столбец, названный именем рецепта (пример: **R1** и **R2** на изображении выше).

В режиме онлайн рецепт может быть изменен или любым соответственно сконфигурированным элементом визуализации (конфигурация входа выполняет команду), или при помощи надлежащих методов функционального блока `RecipeManCommands` библиотеки Recipe_Management.library.

Поддерживаются следующие методы:
- `ReadRecipe`: Текущие значения переменных приняты в рецепт.
- `WriteRecipe`: Рецепт записан в переменные.
- `SaveRecipe`: Рецепт сохранен в стандартный файл рецепта.
- `LoadRecipe`: Рецепт загружен из стандартного файла рецепта.
- `CreateRecipe`: Создается новый рецепт в определении рецепта.
- `DeleteRecipe`: Существующий рецепт удаляется из определения рецепта.

Смотрите в последующих параграфах, как рецепты ведут себя в определенных онлайновых состояниях. Наиболее успешная практика – установить опцию **Save changes torecipes automatically (Сохранять изменения в рецептах автоматически)** (чтобы получить обычное поведение управления рецептом)

Рецепт

Вы можете добавить или удалить рецепт оффлайн или онлайн. В режиме офлайн используйте команды **Add a new recipe***(см. SoMachine, Menu Commands, Online Help)*, и **Remove recipes** *(см. SoMachine, Menu Commands, Online Help)* в редакторе менеджера рецептов. В режиме онлайн или сконфигурируйте ввод соответственно сконфигурированного элемента визуализации, или используйте надлежащие методы функционального блока `RecipeManCommands` библиотеки Recipe_Management.library.

При добавлении рецепта следующий столбец будет добавлен позади самого правого столбца, названного именем рецепта (см. изображение редактора вида определения рецепта). Поля столбца рецепта могут быть заполнены надлежащими значениями. Таким образом, для того же набора переменных могут быть подготовлены различные наборы значений в определенных рецептах.

Использование рецептов в режиме онлайн

Рецепты могут быть обработаны (создание, чтение, запись, сохранение, загрузка, удаление) при помощи методов функционального блока `RecipeManCommands`, предоставленного библиотекой Recipe_Management.libray, в коде приложения или через вводы на элементах визуализации.

Обработка рецепта в режиме онлайн, если активировано **Save changes to recipes automatically**:

Действия	Рецепты, определенные в проекте	Рецепты, создаваемые во время работы
Тёплый перезапуск в онлайн **Холодный перезапуск онлайн** **Download (Загрузка)**	Рецепты всех определений рецепта установлены со значениями из текущего проекта.	Динамически создаваемые рецепты остаются неизменными.
Online Reset Origin (Перезапуск онлайн в исходное состояние)	Приложение будет удалено из контроллера. Если впоследствии будет сделана новая загрузка, рецепты будут восстановлены как на **Online Reset Warm**.	
Выключение и перезапуск контроллера	После перезапуска рецепты перезагружены из автоматически создаваемых файлов. Таким образом будет восстановлено состояние перед отключением.	
Online Change (Изменение онлайн)	Значения рецепта остаются неизменными. Во время работы рецепт может быть изменен только командами функционального блока `RecipeManCommands`.	
Stop (Остановка)	При остановке/запуске контроллера рецепты остаются неизменными.	

Обработка рецепта в режиме онлайн, если **Save changes to recipes automatically** НЕактивировано:

Действия	Рецепты, определенные в проекте	Рецепты, создаваемые во время работы
Тёплый перезапуск в онлайн **Холодный перезапуск онлайн** **Download (Загрузка)**	Рецепты всех определений рецептов установлены со значениями из текущего проекта. Однако они установлены только в памяти. Чтобы сохранить рецепт в файле, команда сохранения должна использоваться явно.	Динамически создаваемые рецепты теряются.
Online Reset Origin (Перезапуск онлайн в исходное состояние)	Приложение будет удалено из контроллера. Если впоследствии будет сделана новая загрузка, рецепты будут восстановлены.	Динамически создаваемые рецепты теряются.
Выключение и перезапуск контроллера	После перезапуска рецепты перезагружены из начальных значений, которые были созданы при загрузке значений вне проекта. Таким образом, состояние перед отключением не будет восстановлено.	
Online Change (Изменение онлайн)	Значения рецепта остаются неизменными. В течение времени выполнения рецепт может быть изменен только командами функционального блока `RecipeManCommands`.	
Stop (Остановка)	При остановке/запуске контроллера рецепты остаются неизменными.	

Дополнительная информация:

- Относительно сохранения рецептов в файлах, которые перезагружаются при перезапуске приложения, обращайтесь к описанию *Редактор Менеджера рецептов, Вкладка Storage (см. стр.433)*.
- Для описания определенных методов `RecipeManCommands` *(см. стр. 439)*, обратитесь к документации в библиотеке.
- Для надлежащей конфигурации ввода элемента визуализации обратитесь к странице справки (категория **Input→execute command**).

С рецептами возможны следующие действия:

Действия	**Описание**
Создание рецепта (= **Add a new recipe**)	Новый рецепт будет создаваться в указанном определении рецепта.
Чтение рецепта	Текущие значения переменных указанного определения рецепта будут считаны из контроллера и записаны в указанный рецепт. Это означает, что значения будут сохранены неявно (в файле на контроллере). Они будут также сразу наблюдаться в таблице определения рецепта в **Recipe Manager**. Другими словами, рецепт, которым управляют в **Recipe Manager**, обновляется фактическими значениями из контроллера.
Запись рецепта	Значения данного рецепта, которые видимы в менеджере рецептов, будут записаны в переменные на контроллере.
Save Recipe **(Сохранение рецепта)**	Значения указанного рецепта будут записаны в файл с расширением **.txtrecipe*, имя которого Вы должны определить. С этой целью стандартное диалоговое окно откроется для того, чтобы сохранить файл в локальной файловой системе. **ПРИМЕЧАНИЕ**: неявно используемые файлы рецепта, необходимые как буфер для чтения и записи значений рецепта, не могут быть перезаписаны. Это означает, что имя для нового файла рецепта должно отличаться от <recipename>.<recipedefinitionname>.txtrecipe.
Load Recipe **(Загрузка рецепта)**	Рецепт, который был сохранен в файле (см. описание **Сохранение рецепта**) может быть перезагружен из этого файла. С этой целью откроется стандартное диалоговое окно для просмотра файла. Фильтр автоматически установлен на расширение **.txtrecipe*. После перезагрузки файла значения рецепта будут соответственно обновлены в менеджере рецептов.
Удаление рецепта (= **Remove recipe**)	Указанный рецепт будет удален из определения рецептов.
Изменение рецепта	Значение переменных проекта может быть изменено. Со следующим действием записи рецепта надлежащие переменные проекта записаны с новыми значениями.

Команды RecipeMan

Обзор

При вызове команды рецепта будет выполнен внутренний доступ к данным. В зависимости от типа устройства это займет несколько миллисекунд. Проверьте, что эти вызовы не выполняются задачей со сконфигурированным сторожевым таймером или задачей в реальном времени. Предполагайте, что опция **Save changes to recipes automatically** также будет выполнять доступ к файлу с каждым изменением рецепта. Деактивируйте эту опцию, если сохранение рецепта инициируется приложением.

Возвращаемые значения

Для команд рецепта возможны следующие возвращаемые значения:

Возвращаемое значение	Описание
ERR_NO_RECIPE_MANAGER_SET	Нет менеджера рецептов, доступного на контроллере.
ERR_RECIPE_DEFINITION_NOT_FOUND	Определение рецепта не существует.
ERR_RECIPE_ALREADY_EXIST	Рецепт уже существует в определении рецепта.
ERR_RECIPE_NOT_FOUND	Рецепт не существует в определении рецепта.
ERR_RECIPE_FILE_NOT_FOUND	Файл рецепта не существует.
ERR_RECIPE_MISMATCH	Содержание файла рецепта не соответствует текущему рецепту. **ПРИМЕЧАНИЕ**: Это возвращаемое значение генерируется толькотогда, когда тип сохранения текстовый и когда имя переменной в файле не соответствует имени переменной в определении рецепта. Файл рецепта не загружен.
ERR_RECIPE_SAVE_ERR	Файл рецепта не мог быть открыт с доступом для записи
ERR_FAILED	Операция была неуспешной.
ERR_OK	Операция была успешной.

CreateRecipe

Этот метод создает новый рецепт в указанном определении рецепта и впоследствии читает текущие значения контроллера в новый рецепт. В конце новый рецепт сохраняется в файле по умолчанию.

Параметр	Описание
RecipeDefinitionName:	имя определения рецепта
RecipeName:	имя рецепта

Возвращаемые значения *(см. стр. 439)*:
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_ALREADY_EXIST, ERR_FAILED, ERR_OK

CreateRecipeNoSave

Этот метод создает новый рецепт в указанном определении рецепта и впоследствии читает текущие значения контроллера в новый рецепт.

Параметр	Описание
RecipeDefinitionName:	Имя определения рецепта
RecipeName:	Имя рецепта

Возвращаемые значения *(см. стр. 439)*:
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_ NOT_FOUND, ERR_FAILED, ERR_OK

DeleteRecipe

Этот метод удаляет рецепт из определения рецепта.

Параметр	Описание
RecipeDefinitionName:	Имя определения рецепта
RecipeName:	Имя рецепта

Возвращаемые значения *(см. стр. 439)*:
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_ NOT_FOUND, ERR_FAILED, ERR_OK

DeleteRecipeFile

Этот метод удаляет стандартный файл рецепта из рецепта.

Параметр	Описание
RecipeDefinitionName:	Имя определения рецепта
RecipeName:	Имя рецепта

Возвращаемые значения *(см. стр. 439)*:
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_ NOT_FOUND, ERR_RECIPE_FILE_NOT_FOUND, ERR_OK

LoadAndWriteRecipe

Этот метод загружает рецепт из стандартного файла рецепта и впоследствии пишет рецепт в переменные контроллера.

Параметр	Описание
RecipeDefinitionName:	Имя определения рецепта
RecipeName:	Имя рецепта

Возвращаемые значения *(см. стр. 439)*:
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_ NOT_FOUND, ERR_RECIPE_FILE_NOT_FOUND, ERR_RECIPE_MISMATCH, ERR_FAILED, ERR_OK

LoadFromAndWriteRecipe

Этот метод загружает рецепт из указанного файла рецепта и впоследствии пишет рецепт в переменные контроллера.

Параметр	Описание
RecipeDefinitionName:	Имя определения рецепта
RecipeName:	Имя рецепта
FileName:	Имя файла

Возвращаемые значения *(см. стр. 439)*:
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_ NOT_FOUND, ERR_RECIPE_FILE_NOT_FOUND, ERR_RECIPE_MISMATCH, ERR_FAILED, ERR_OK

LoadRecipe

Этот метод загружает рецепт из стандартного файла рецепта. Стандартное имя файла рецепта <recipe>.<recipe definition>.<recipeextension>.

Параметр	Описание
RecipeDefinitionName:	Имя определения рецепта
RecipeName:	Имя рецепта

Возвращаемые значения *(см. стр. 439)*:
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_ NOT_FOUND, ERR_RECIPE_FILE_NOT_FOUND, ERR_RECIPE_MISMATCH, ERR_FAILED, ERR_OK

ReadAndSaveRecipe

Этот метод читает текущие значения контроллера в рецепте и впоследствии сохранит рецепт в стандартном файле рецепта.

Параметр	Описание
RecipeDefinitionName:	Имя определения рецепта
RecipeName:	Имя рецепта

Возвращаемые значения *(см. стр. 439)*:
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_ NOT_FOUND, ERR_RECIPE_SAVE_ERR, ERR_FAILED, ERR_OK

ReadAndSaveRecipeAs

Этот метод читает текущие значения контроллера в рецепте и впоследствии сохранит рецепт в указанном файле рецепта. Содержание существующего файла будет перезаписано.

Параметр	Описание
RecipeDefinitionName:	Имя определения рецепта
RecipeName:	Имя рецепта
FileName:	Имя файла

Возвращаемые значения *(см. стр. 439)*:
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_ NOT_FOUND, ERR_RECIPE_SAVE_ERR, ERR_FAILED, ERR_OK

SaveRecipe

Этот метод сохраняет рецепт в стандартном файле рецепта. Содержание существующего файла будет перезаписано. Стандартное имя файла рецепта - <recipe>.<recipedefinition>.<recipeextension>.

Параметр	Описание
RecipeDefinitionName:	Имя определения рецепта
RecipeName:	Имя рецепта

Возвращаемые значения *(см. стр. 439)*:
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_ NOT_FOUND, ERR_RECIPE_SAVE_ERR, ERR_FAILED, ERR_OK

ReadRecipe

Этот метод читает текущие значения контроллера в рецепт.

Параметр	Описание
RecipeDefinitionName:	Имя определения рецепта
RecipeName:	Имя рецепта

Возвращаемые значения *(см. стр. 439)*:
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_ NOT_FOUND, ERR_FAILED, ERR_OK

WriteRecipe

Этот метод пишет рецепт в переменные контроллера.

Параметр	Описание
RecipeDefinitionName:	Имя определения рецепта
RecipeName:	Имя рецепта

Возвращаемые значения *(см. стр. 439)*:
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_ NOT_FOUND, ERR_FAILED, ERR_OK

ReloadRecipes

Этот метод перезагружает список рецептов из файловой системы.

Параметр	Описание
RecipeDefinitionName:	Имя определения рецепта

Возвращаемые значения *(см. стр. 439)*:
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_FAILED, ERR_OK

GetRecipeCount

Этот метод возвращает количество рецептов из соответствующего определения рецепта.

Параметр	Описание
RecipeDefinitionName:	Имя определения рецепта

Возвращаемые значения:–1: если определение рецепта не найдено.

GetRecipeNames

Этот метод возвращает имена рецепта из соответствующего определения рецепта.

Параметр	Описание
`RecipeDefinitionName`:	Имя определения рецепта
`pStrings`:	строки, где значения рецепта должны быть сохранены
`iSize`:	размер массива строк
`iStartIndex`:	Стартовый индекс может использоваться для функции прокрутки

Возвращаемые значения *(см. стр. 439)*:
`ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_FAILED, ERR_OK`

Пример:

Есть 50 рецептов. Чтобы составить таблицу, которая показывает 10 имен рецепта за один раз, определите массив последовательности:

```
strArr: ARRAY[0..9] OF STRING;
```

Соотносясь с `iStartIndex`, имена рецепта могут быть считаны из определенной области.

```
iStartIndex := 0;
```

Имена 0... 9 возвращаются.

```
iStartIndex := 20;
```

Имена 20... 29 возвращаются. В этом примере:

```
iSize := 10;
```

GetRecipeValues

Этот метод возвращает значения переменных рецепта из соответствующего рецепта.

Параметр	Описание
`RecipeDefinitionName`:	Имя определения рецепта
`RecipeName`	Имя рецепта
`pStrings`:	строки, где значения рецепта должны быть сохранены
`iSize`:	размер массива строк
`iStartIndex`:	Стартовый индекс может использоваться для функции прокрутки
`iStringLength`:	длина строки в массиве

Возвращаемые значения *(см. стр. 439)*:
`ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_ NOT_FOUND, ERR_FAILED, ERR_OK`

Пример:

Есть 50 рецептов. Чтобы составить таблицу, которая показывает 10 имен рецепта за один раз, определите массив последовательности:

```
strArr: ARRAY[0..9] OFSTRING;
```

Соотносясь с `iStartIndex`, имена рецепта могут быть считаны из определенной области.

```
iStartIndex := 0;
```

Значения 0... 9 возвращаются.

```
iStartIndex := 20;
```

Значения 20... 29 возвращаются. В этом примере:

```
iStringLength := 80;
iSize := 10;
```

GetRecipeVariableNames

Этот метод возвращает имена переменных соответствующего рецепта.

Параметр	Описание
RecipeDefinitionName:	Имя определения рецепта
RecipeName	Имя рецепта
pStrings:	строки, где значения рецепта должны быть сохранены
iSize:	Размер массива строк
iStartIndex:	Стартовый индекс может использоваться для функции прокрутки

Возвращаемые значения *(см. стр. 439)*:
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_ NOT_FOUND, ERR_FAILED, ERR_OK

Пример:

Есть 50 рецептов. Чтобы составить таблицу, которая показывает 10 имен рецепта за один раз, определите массив последовательности:

```
strArr: ARRAY[0..9] OFSTRING;
```

Соотносясь с iStartIndex, имена рецепта могут быть считаны из определенной области.

```
iStartIndex := 0;
```

Имена 0... 9 возвращаются.

```
iStartIndex := 20;
```

Имена 20... 29 возвращаются. В этом примере:

```
iSize := 10;
```

SetRecipeValues

Этот метод устанавливает значения рецепта в соответствующий рецепт.

Параметр	Описание
RecipeDefinitionName:	Имя определения рецепта
RecipeName	Имя рецепта
pStrings:	строки, где значения рецепта должны быть сохранены
iSize:	размер массива строк
iStartIndex:	Стартовый индекс может использоваться для функции прокрутки

Возвращаемые значения *(см. стр. 439)*:
ERR_NO_RECIPE_MANAGER_SET, ERR_RECIPE_DEFINITION_NOT_FOUND, ERR_RECIPE_ NOT_FOUND, ERR_FAILED, ERR_OK

Пример:

Есть 50 рецептов. Чтобы составить таблицу, которая показывает 10 имен рецепта за один раз, определите массив последовательности:

```
strArr: ARRAY[0..9] OF STRING;
```

Соответствуя iStartIndex, имена рецепта могут быть считаны из определенной области.

```
iStartIndex := 0;
```

Значения 0... 9 установлены.

```
iStartIndex := 20;
```

Значения 20... 29 установлены. В этом примере:

```
iStringLength := 80;
iSize := 10;
```

GetLastError

Этот метод возвращает последнюю обнаруженную ошибку предыдущих операций.

Возвращаемые значения *(см. стр. 439)*:
ERR_NO_RECIPE_MANAGER_SET, ERR_OK

ResetLastError

Этот метод сбрасывает последнюю обнаруженную ошибку.
Возвращаемые значения *(см. стр. 439)*:
ERR_NO_RECIPE_MANAGER_SET, ERR_OK

Глава 24

Редактор трассировки

Содержание этой главы

Эта глава содержит следующие темы:

Раздел	Тема	Страница
24.1	Объект трассировки	450
24.2	Конфигурация трассировки	456
24.3	Редактор трассировки в режиме онлайн	470
24.4	Операции клавиатуры для графиков трассировки	471

Раздел 24.1

Объект трассировки

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Основы трассировки	451
Создание объекта трассировки	453

Основы трассировки

Функциональность трассировки

Функциональность трассировки позволяет Вам получать последовательность значений переменных на контроллере за определенное время, подобно сэмплированию (периодическому опросу) цифрового осциллографа. Кроме того, Вы можете установить триггер управления получением данных с входными (триггерными) сигналами. Значения переменных трассировки постоянно пишутся в буфер SoMachine указанного размера. Они могут наблюдаться в форме двухмерного графика, построенного в функции времени.

Способ трассировки данных

Управление трассировкой данных выполняется 2 различными способами:

- или любым кодом IEC, сгенерированным объектом трассировки, и загруженным в контроллер дочерним приложением трассировки.
- или в компоненте `CmpTraceMgr` (называемой также **Trace Manager** - менеджером трассировки).

То, какие данные регистрируются, определено записью в целевых параметрах настройки (**trace** → **trace manager**). Менеджер по трассировке имеет расширенную функциональность. Он позволяет Вам:

- сконфигурировать и проследить параметры системы управления, такие как кривая температуры процессора или батареи. Для получения дополнительной информации обратитесь к параметрам настройки переменных *(см. стр. 457)*, и к параметрам настройки записи (триггера) *(см. стр. 460)*.
- Считать трассировки устройства вывода, такие как трассировка электрического тока привода. Для получения дополнительной информации обратитесь к описанию команды **UploadTrace***(см. SoMachine, Menu Commands, Online Help)*,
- Проводить трассировку системных переменных других системных компонентов.

Кроме того, доступна дополнительная команда **Online List** *(см. SoMachine, Menu Commands, Online Help)*

ПРИМЕЧАНИЕ:

- Если трассировка используется в визуализации, параметры устройства не могут прослеживаться или использоваться для триггера.
- Пороговый уровень не может быть установлен как IEC выражение, поддерживаются только литералы и константы.
- Условие записи не может быть установлено как IEC выражение типа BOOL, поддерживаются только переменные.
- Если свойство прослеживается или используется для триггера, оно должно быть аннотировано атрибутом мониторинга *(см. стр. 556)* в IEC объявлении.

Конфигурация

Сконфигурируйте данные трассировки, а также параметры настройки дисплея данных трассировки в **Configuration**. Это обеспечивает команды для доступа к диалоговым окнам конфигурации. Несколько переменных могут быть отслежены и выведены на экран одновременно в различных видах, таких как многоканальный режим. Запись производит трассировку переменных с различными параметрами настройки триггеров в их собственном объекте трассировки. Вы можете создать любое количество объектов трассировки.

Дерево инструментов **Toolstree** с несколькими объектами трассировки

Команды для изменения параметров настройки дисплея описаны в параграфе *Свойства* *(см. стр. 454)*. Доступны функциональности изменение масштаба и курсора, а также команды для выполнения трассировки так, чтобы график мог быть сжат или растянут.

Чтобы интегрировать считывание трассировки в визуализацию, используйте элемент визуализации **Trace**.

Создание объекта трассировки

Обзор

Чтобы вставить объект трассировки в **Toolstree**, выберите узел **Application**, щелкните по кнопке зеленого плюса и выполните команду **Add other objects** → **Trace...**. Дважды щелкните по узлу **Trace** в **Toolstree**, чтобы открыть редактор трассировки.

Configuration Конфигурация

Новая создаваемая трассировка с контекстным меню

Трассировка содержит, по крайней мере, одну переменную, которая регистрируется.

В области дерева трассировки в правой части окна выведены на экран сконфигурированные переменные трассировки. По умолчанию переменные трассировки выведены на экран с их полным путём установки экземпляра.

Установите флажок **Hide instance paths**, чтобы скрыть путь экземпляра. Чтобы вывести на экран этот флажок, нажмите кнопку стрелки в правом верхнем углу области дерева трассировки.

Чтобы сконфигурировать или изменить настройки трассировки, используйте команды контекстного меню в области дерева трассировки:

- **AddVariable...**: Открывает диалоговое окно **Trace Configuration** с **VariableSettings** (*см. стр. 457*),
- **DeleteVariable**: Удаляет выбранную переменную. Доступна только, если существует, по крайней мере, одна переменная трассировки.
- **Visible**: Эта команда делает выбранную переменную видимой. Доступна только, если существует, по крайней мере, одна переменная трассировки.
- **Appearance ...**: Открывается диалоговое окно **Edit Appearance** *(см. стр. 465)*. Оно позволяет Вам конфигурировать появление графика и системы координат. Эта команда отображается серым, пока конфигурация не загружена.
- **Configuration...**: Открывает диалоговое окно **Trace Configuration** с **RecordSettings** *(см. стр. 460)*.

Свойства

Для выполнения трассировки используйте следующие команды:
- **Add variable (добавить переменную)** *(см. SoMachine, Menu Commands, Online Help)*
- **Download Trace (загрузить трассировку)** *(см. SoMachine, Menu Commands, Online Help)*
- **Start/Stop Trace (Старт/Стоп трассировки)** *(см. SoMachine, Menu Commands, Online Help)*
- **Reset Trigger (сброс триггера)** *(см. SoMachine, Menu Commands, Online Help)*

Для настройки вида графиков используйте следующие команды:
- **Cursor (курсор)** (см. *SoMachine, Menu Commands, Online Help)*
- **Mouse Zooming (изменение масштаба мышью)***(см. SoMachine, Menu Commands, Online Help)*
- **Reset View (сброс вида)***(см. SoMachine, Menu Commands, Online Help)*
- **Auto Fit (автоматическая подгонка масштаба)** (см. *SoMachine, Menu Commands, Online Help)*
- **Compress (сжатие)** (см. *SoMachine, Menu Commands, Online Help)*
- **Stretch (растяжка)** (см. *SoMachine, Menu Commands, Online Help)*
- **MultiChannel (многоканальность)** *(см. SoMachine, Menu Commands, Online Help)*
- Для получения дополнительной информации обратитесь к главе (*см. стр. 471*) *Операции клавиатуры для графиков трассировки.*

Для доступа к трассировкам, сохраненным в системе во время выполнения, используйте следующие команды:
- **Online List (список онлайн)** *(см. SoMachine, Menu Commands, Online Help)*
- **Upload Trace (выгрузка трассировки)** *(см. SoMachine, Menu Commands, OnlineHelp)*

Для доступа к трассировкам, сохраненным на диске, используйте следующие команды:
- **SaveTrace... (сохранить трассировку как…)***(см. SoMachine, Menu Commands, Online Help)*
- **LoadTrace... (загрузить трассировку)***(см. SoMachine, Menu Commands, Online Help)*
- **Export symbolic trace config (экспорт конфигурации символьной трассировки)** *(см.SoMachine, Menu Commands, Online Help)*

Начало работы

Чтобы запустить трассировку в режиме онлайн, загрузите конфигурацию трассировки на контроллер, выполнив команду **Download Trace**. Графики переменных трассировки будут выведены на экран в окне редактора трассировки, где Вы можете сохранить их во внешнем файле. Этот файл может быть перезагружен в редактор. Также обратитесь к главе *Редактор трассировки в режиме онлайн* (*см. стр. 470*).

Шаг	Действие
1	Войдите в систему и запустите связанное приложение. **Результат**: Приложение работает на контроллере.
2	Загрузите трассировку **Результат:** графики трассировки сразу отображаются согласно конфигурации трассировки.
3	Расположение графиков трассировки, сохранение данных трассировки, остановка/запуск трассировки.

Пример

Редактор трассировки показывает пример трассировки в режиме онлайн. Четыре переменные для отображения были выбраны в дереве переменных в правой части диалогового окна.

Трассировка в режиме онлайн.

Раздел 24.2

Конфигурация трассировки

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Параметры настройки переменных	457
Параметры настройки записи	460
Расширенные параметры настройки трассировки	463
Редактирование изображения	465
Изображение Y-оси	469

Параметры настройки переменных

Обзор

Когда Вы выбираете переменную трассировки в дереве трассировки открывается диалоговое окно **Trace Configuration** с параметрами настройки переменных **Variable Settings**. Это позволяет Вам сконфигурировать, какие переменные должны быть прослежены и как они выведены на экран.

Диалоговое окно **Trace Configuration** с параметрами настройки переменных (**Variable Settings)**

Переменные трассировки выведены на экран в левой части окна в древовидной структуре. Главный узел назван именем трассировки.

Добавление и удаление переменной трассировки

Для добавления переменной к дереву трассировки или удалению ее, используйте приведенные ниже команды дерева трассировки:

Команда	Описание
Add Variable (добавить переменную)	создает анонимную запись в дереве трассировки. В правой части диалогового окна параметры настройки новой переменной готовы к конфигурации.
Delete Variable (удалить переменную)	удаляет выбранную переменную со связанной конфигурацией.

Установка и изменение настроек переменных

Для выбора параметров настройки переменных выберите желаемую переменную в дереве трассировки. Текущие параметры настройки будут отражены в правой части окна конфигурации трассировки. Чтобы позже изменить параметры настройки переменной, выберите запись переменной в дереве трассировки и снова используйте диалоговое окно **Variable Settings**.

Параметр		Описание
Variable (Переменная)		Введите имя (путь) для определения сигнала, который будет прослежен. Допустимый сигнал - IEC переменная, свойство, ссылка, содержание указателя или элемент матрицы из приложения. Разрешённые типы - все основные типы IEC, кроме STRING, WSTRING или ARRAY. Позволены также те перечисления, чей основной тип не STRING,WSTRING или ARRAY. Щелкните кнопку **...**, чтобы открыть ассистент ввода, который позволяет Вам получать допустимый элемент. Контроллеры, которые поддерживают трассировку параметра, предоставляют список, если Вы нажимаете **Variable:** параметр. Если Вы хотите проследить параметр устройства, выберите элемент **Parameter** из этого списка. Тогда Вы можете найти его с помощью ассистента ввода. Отредактируйте или проверьте текущие параметры настройки переменных. Параметры устройства поддерживаются, только если используется компонент `CmpTraceMgr`. Если параметры устройства используются для переменных трассировки (или для триггера), опцию **Generate Trace POU for visualization** (генерация POU трассировки для визуализации) активировать невозможно. **ПРИМЕЧАНИЕ**: Если для трассировки используется `CmpTraceMgr`, переменная Property (свойство) (*см. стр. 161*), которая используется как переменная трассировки (или триггер), должна получить атрибут компилятора Attribute Monitoring (*см. стр. 556*).
Graph color		Выберите цвет из списка выбора цветов, в котором будет выведена на экран кривая трассировки для переменной.
Line type		Определите способ, которым выборки будут соединены в графике. Используйте **Line** (линия) для больших объемов данных. Это также значение по умолчанию.
Тип линии	**Line**	Выборки соединены линией (значение по умолчанию).
	Step	Выборки соединены в форме лестницы. Таким образом, горизонтальная строка к метке времени следующей выборки сопровождается вертикальной строкой к значению следующей выборки.
	None	Выборки не соединены.
Point type		Определите, как сами значения будут нарисованы в графике.
Тип точек	**Dot**	Выборки изображены как точки (значение по умолчанию).
	Cross	Выборки изображены как кресты.
	None	Выборки не отображаются.
Activate Minimum Warning		Если эта опция активирована, участок графика трассировки будет отображён в цвете, определенном в **Warning minimum color**, как только переменная выходит за значение, определенное в **Criticall owerlimit**.
Critical lower limit		Если значение переменной меньше введенного здесь и активна опция **Activate minimum warning**, значения кривой изменяются на следующий указанный цвет.
Warning minimum color		Значение цвета для активированного нижнего предела.

Параметр	Описание
Activate Maximum Warning	Если эта опция активирована, участок графика трассировки будет выведен на экран в цвете, определенном в **Warning maximum color**, как только переменная превышает значение, определенное в **Critical upper limit**.
Critical upper limit	Если значение переменной больше введенного здесь и активна опция **Activate maximum warning**, значения кривой изменяются на следующий указанный цвет.
Warning maximum color	Значение цвета для активированного верхнего предела.
Appearance...	Открывает диалоговое окно **Appearance оси Y**. Оно позволяет Вам устанавливать отображение окна трассировки для сконфигурированной в настоящее время оси Y (цвета и поведение прокрутки) для каждой переменной в ее собственном стиле. Эти параметры настройки используются, когда изображение трассировки отображается в многоканальном представлении.

Выбор нескольких переменных

При помощи сочетаний клавиш SHIFT + щелчок мышью или CTRL + щелчок мышью, Вы можете выбрать несколько переменных для редактирования (мультивыбор). Тогда изменения в диалоговом окне **Variable Settings** применяются ко всем выбранным переменным. То же может быть сделано с SHIFT + СТРЕЛКА/ВВЕРХ или CTRL + СТРЕЛКА/ВНИЗ.

Мультивыбор в диалоговом окне **Trace Configuration.**

Параметры настройки записи

Обзор

Диалоговое окно **Trace Configuration** с параметрами настройки **Record Settings** открывается, если Вы выполняете команду **Configuration...** или если Вы дважды щелкаете по имени трассировки на вершине дерева трассировки. Команда конфигурации также доступна в контекстном меню дерева трассировки на правой части основного редактора окна трассировки.

ПРИМЕЧАНИЕ: параметры настройки, завершенные в диалоговом окне **Trace Configuration** с параметрами настройки записи (**Record Settings)**, верны для всех переменных графика трассировки.

Основы Триггера запуска

В большинстве ситуаций нежелательно, чтобы трассировка и отображение входных сигналов стартовали в случайные моменты, такие как сразу после предыдущего измерения, или когда пользователь нажимает кнопку запуска. В большинстве случаев предпочтительно, чтобы трассировка делалась, когда триггер был включён после сконфигурированного количества записей (пост-триггер). Это называется запуском и определяется здесь.

Используются следующие пути для инициирования записи входных сигналов:

- Конфигурируя переменную, переключающую триггер,
- Конфигурируя условие записи,
- или используя оба способа

Установка и изменение параметров настройки Record (Trigger)

Диалоговое окно **Trace Configuration** с параметрами настройки **Record Settings**.

Параметр		Описание
Enable Trigger		Установите флажок, чтобы включить триггерную систему. Она может быть включена или выключена независимо от более низких параметров настройки. Если триггерная система отключена, трассировка имеет свободный ход.
Trigger Variable		Назначьте переменную. Определите, какой сигнал будет использоваться в качестве триггера, вводя имя (и путь) сигнала. Допустимый сигнал триггера - IEC переменная, свойство, ссылка, указатель, элемент матрицы из приложения или выражение. Разрешённые типы - все основные типы IEC, кроме STRING, WSTRING или ARRAY. Разрешены также перечисления, чей основной тип не STRING, WSTRING или ARRAY. Содержание указателя - недопустимый сигнал. Щелкните кнопку **...**, чтобы открыть ассистента ввода, который позволяет Вам получать доступ к допустимому элементу. Контроллеры, которые поддерживают использование параметров устройства как триггеров, обеспечивают список, если Вы нажимаете **Trigger Variable**: параметр. Если Вы хотите использовать параметр устройства в качестве триггера, выберите элемент **Trigger Parameter** из этого списка. Откройте **Input Assistant** кнопкой **...** и выберите отслеживаемые параметры. Под **Elements** перечислены параметры, доступные в системе. Вы можете также ввести имена параметров непосредственно или копированием и вставкой (из другой конфигурации) в текстовое поле. Параметры устройства поддерживаются, только если используется менеджер трассировки. **ПРИМЕЧАНИЕ:** Если для трассировки используется `CmpTraceMgr`, переменная Property *(см. стр. 161)*, которая используется в качестве переменной трассировки (или триггера), должна получить атрибут компилятора **Attribute Monitoring***(см. стр. 556)*.
Trigger edge		–
	positive	Событие переключения при нарастающем фронте булевой триггерной переменной. Или как только значение, определенное уровнем **Trigger Level** для аналоговой триггерной переменной, достигнуто при возрастании.
	negative	Событие переключения при убывающем фронте булевой триггерной переменной. Или как только значение, определенное уровнем **Trigger Level** для аналоговой триггерной переменной, достигнуто при убывании.
	both	Событие переключения при условиях, описанных для **positive** и **negative**.
Post Trigger		Введите число записей на сигнал трассировки, которые регистрируются после того, как триггер запущен. Значение по умолчанию: 50 Диапазон:0... (2^{32}-1)
Trigger Level		Введите значение, в какой точке запускается триггер. По **Triggeredge** Вы можете определить, запускается ли он при возрастающем или убывающем фронте триггерной переключаемой переменной. Это должно быть установлено, если и только если аналоговая переменная (переменная с числовым типом, таким как LREAL или INT) используется в качестве триггерной переменной. Непосредственно введите значение. Константа GVL или значение ENUM разрешены, если их тип конвертируем к триггерной переменной. Если используется код IEC, то Вы можете также ввести произвольное выражение типа IEC, которое конвертируемо к триггерной переменной. Значение по умолчанию: – (отсутствует)

Параметр	Описание
Task	Из списка доступных задач выберите ту, где происходит получение входных сигналов.
Record condition	Если Вы хотите запустить запись условием, введите здесь переменную. Если трассировка запущена прежде, например, нажатием кнопки запуска, и переменная, присвоенная здесь, становится TRUE, получение данных запускается, и график трассировки будет выведен на экран. Если используется `CmpTraceMgr`, условие записи должно быть переменной типа BOOL или битового доступа. Содержание указателя – недопустимый элемент. Свойства также поддерживаются. Если используется код IEC, произвольное IEC выражение типа BOOL также может быть введено.
Comment	Введите текст комментария относительно текущей записи.
Resolution	Введите разрешающую способность метки времени трассировки в мс или мкс. Для каждого зарегистрированного сигнала, пара значение и метка времени сохраняются и передаются в систему программирования. Переданные метки времени относительны и относятся к моменту запуска трассировки. Если у задачи трассировки время цикла 1 мс или меньше, рекомендуется метка времени с разрешением в мкс. Эта опция возможна, только когда в контроллере доступен менеджер трассировки.
Create persistent record	Установите эту опцию, если конфигурация трассировки и последнее содержание буферов трассировки RTS должно постоянно сохраняться на целевом устройстве. Эта опция возможна, только когда менеджер трассировки осуществляет трассировку в контроллере.
Appearance...	Открывается диалоговое окно **Appearance of the Y-axis** (*см. стр. 469*). Оно позволяет Вам устанавливать свойства отображение окна трассировки для сконфигурированной в настоящее время записи, такие как оси, цвета и поведение прокрутки.
Advanced...	Нажмите эту кнопку, чтобы открыть диалоговое окно **Advanced Trace Settings** *(см. стр. 463)*. Это позволяет Вам устанавливать некоторые дополнительные параметры настройки для триггера трассировки.

ПРИМЕЧАНИЕ: Если Вы хотите получить и отобразить сигнал трассировки с различной базой времени, Вы должны реализовать конфигурацию записи в отдельном объекте трассировки.

Расширенные параметры настройки трассировки

Обзор

Открывается диалоговое окно **Advanced Trace Settings**, если Вы нажимаете кнопку **Advanced...** в диалоговом окне **Trace Configuration** с параметрами настройки **Record Settings**.

Диалоговое окно **Advanced Trace Settings** расширенных параметров настройки трассировки

Для всех значений, заданных в записях или циклах, связанный отрезок времени показан непосредственно рядом с ним справа (например, **1m39s990ms**). Отрезок времени для буферов включает весь буфер. Если задача не установлена, не циклична или это системная задача, то время цикла задачи неизвестно. В этом случае период не может быть вычислен и не будет выведен на экран с правой стороны.

Описание параметров

Параметр	Описание
Refresh rate (ms) **(Интервал обновления)**	В этот временной интервал полученные пары данных (значение с меткой времени) трассировки, сохранены в буфере редактора трассировки. Диапазон: 150 ms...10000 ms Значение по умолчанию: 500 ms Если используется менеджер трассировки, то пары данных передаются с этим периодом времени от системы реального времени к системе программирования. Если менеджер трассировки не используется, то данные передаются к системе программирования каждые 200 мс.
Trace editor buffer size (samples)	Введите размер буфера трассировки в выборках (записях). Этот буфер должен быть больше чем или равным двойному размеру того, каким может быть runtime buffer. Диапазон:1... 10^7
Measure in every n-th cycle	Выберите временной интервал (в циклах задачи) между получением входного сигнала из списка. Значение по умолчанию и минимальное значение: 1 (это означает измерение в каждом цикле),

Параметр	Описание
Recommended runtime buffer size (samples)	Рекомендуемое количество выборок буфера runtime (времени выполнения) отображается для каждого сигнала трассировки. Это значение вычислено на основе времени цикла задачи, времени обновления и значения **Measure in every n-th cycle**. Это означает, что для каждой переменной трассировки выделяется по одному буферу. **ПРИМЕЧАНИЕ**: размер буфера дан в выборках, и для каждой переменной трассировки создается один буфер.
Override runtime buffer size	Когда выбрана эта опция, используется введенное здесь значение размера runtime buffer вместо значения по умолчанию. Пример: Диапазон: 10... Trace editor buffer size

Редактирование изображения

Обзор

Диалоговое окно **Edit Appearance** открывается, если Вы нажимаете кнопку **Appearance...** в диалоговом окне **Trace Configuration** с **Record Settings**.

Следующие параметры установки определяют координатную систему и её X/Y-оси. Параметры установки для Y-оси используются, когда изображение трассировки отображается в одноканальном виде. В многоканальном виде установки производятся в диалоговом окне **Appearance of the Y-axis**.

Уставки для X/Y-оси (заданные в левой части) координатной системы немедленно принимаются в координатной системе, отображаемой справа.

Вы можете управлять уставками с помощью следующих кнопок:

Кнопка	Описание
Reset	По этой команде вид изображения устанавливается на своё значение по умолчанию.
Use as default	По этой команде текущий вид изображения устанавливается как вид по умолчанию. Он может быть использован, когда конфигурируются новая трассировка или переменная.

Вкладка X-axis

Вкладка X-axis диалогового окна **Edit Appearance**

Параметр		Описание
Display mode (Режим изображения)		Выбор режима отображения.
	Auto	Если эта опция активирована, ось времени автоматически масштабируется в соответствии с содержанием буфера редактора трассировки *(см. стр.463)*. Текущее содержимое буфера трассировки отображается на графике. Последующих установок не требуется.
	Fixed length	Если эта опция активирована, отображаемый интервал оси времени имеет фиксированную длину. Определите эту длину параметром **Length**. Масштаб также подстраивается к длине. График автоматически прокручивается в видимом диапазоне. Следовательно, интервал времени со сконфигурированной длиной и связанные с этим интервалом последние данные показываются на графике. Но только столько значений, сколько были зарегистрированы. Как только новые данные регистрируются, прокручивается изображение.
	Fixed	Если эта опция активирована, отображаемый интервал оси времени определяется минимальным и максимальным значением.
Minimum		Это значение определяет минимальное отображаемое значение по оси времени. [1]
Maximum		Это значение определяет максимальное отображаемое значение по оси времени. [1]
Length		Это значение определяет длину отображаемого интервала по оси времени. [1]
Grid		Если эта опция активирована, будет отображаться сетка. Выберите цвет линий сетки из списка
Tick marks		–
	Fixed spacing	Установите этот флажок, чтобы масштабировать ось в определённых интервалах.
	Distance	Введите положительный интервал. Ось X–ось времени со значением по умолчанию 1 сек.
	Subdivision	Введите обоснованное число дополнительных делений для каждого основного интервала.
Font		Открывает стандартное диалоговое окно для определения шрифта в окне трассировки.

[1]Для вводимых значений времени не требуется префикса #, как это требует код IEC. Возможные вводимые значения, например, **2 s**,**1 ms** или **1 m20s14ms**. Используйте us для микросекунд, например,**122 ms500us**. Полезные значения также зависят от разрешающей способности по оси времени.

Вкладка Y-axis

Вкладка Y-axis диалогового окна **Edit Appearance**

Параметр		Описание
Display mode		Выбор режима отображения.
	Auto	Если эта опция активирована, ось Y автоматически масштабируется в соответствии с зарегистрированными значениями. Ввода последующих установок не требуется.
	Fixed	Если эта опция активирована, отображаемый интервал оси Y определяется минимальным и максимальным значением.
Minimum		Это значение определяет минимальное отображаемое значение по оси Y.
Maximum		Это значение определяет максимальное отображаемое значение по оси Y.
Grid		Если эта опция активирована, будет отображаться сетка. Выберите цвет линий сетки из списка.
Description		Если эта опция активирована, ось Y маркируется текстом, введённым в поле, следующее за этим.
Tick marks		–
	Fixed spacing	Установите этот флажок, чтобы масштабировать ось в определённых интервалах.
	Distance	Введите положительный интервал. Значение по умолчанию: 1
	Subdivision	Введите число от 1 до 10 дополнительных делений для каждого основного интервала.
Font		Нажмите эту кнопку для открытия стандартного диалогового окна для определения шрифта для окна трассировки.

Параметры координатной системы

Параметр	Описание
Backcolor	Выберите цвет фона для координатной системы из списка. Он используется до тех пор, пока график не выбран в окне трассировки.
Backcolor on Selection	Выберите цвет фона для координатной системы из списка. Он используется до тех пор, пока график выбран в окне трассировки.

Изображение оси Y

Обзор

Диалоговое окно **Appearance of the Y-axis** открывается, если Вы нажимаете кнопку **Appearance...** в диалоговом окне **Trace Configuration** с опцией **Variable Settings**.

Следующие уставки определяют вид оси Y. Они используются, когда график трассировки отображается в мультиканальном виде.

Диалоговое окно **Appearance of the Y-axis**

Параметр		Описание
Display mode		Выбор режима отображения.
	Auto	Если эта опция активирована, ось Y автоматически масштабируется в соответствии с зарегистрированными значениями. Ввода последующих установок не требуется.
	Fixed	Если эта опция активирована, отображаемый интервал оси Y определяется минимальным и максимальным значением.
Minimum		Это значение определяет минимальное отображаемое значение по оси Y.
Maximum		Это значение определяет максимальное отображаемое значение по оси Y.
Grid		Если эта опция активирована, будет отображаться сетка. Выберите цвет линий сетки из списка.
Description		Если эта опция активирована, ось Y маркируется текстом, введённым в поле, следующее за этим.
Tick marks		–
	Fixed spacing	Установите этот флажок, чтобы масштабировать ось в определённых интервалах.
	Distance	Введите положительный интервал. Значение по умолчанию: 1
	Subdivision	Введите число дополнительных делений для каждого основного интервала.
Font		Открывает стандартное диалоговое окно для определения шрифта для окна трассировки.

Раздел 24.3
Редактор трассировки в режиме онлайн

Редактор трассировки в режиме онлайн

Обзор

Если трассировка работает в устройстве, это индицируется в диалоговом окне **Online List** *(см. SoMachine, Menu Commands, Online Help)*.

Загрузка трассировки

Чтобы запустить трассировку в режиме онлайн, явно загрузите трассировку в контроллер командой меню **Trace** → **Download Trace** *(см. SoMachine, Menu Commands, Online Help)* пока приложение в контроллере открыто. Графики сигналов трассировки будут отображаться в окне редактора трассировки.

Пока производятся подключение и отключение от приложения (login/logout) без его изменения трассировки работают без новой загрузки.

Если код приложения изменился, тогда то, что происходит с трассировками, зависит от режима подключения:

- **Login with online change** (подключение с онлайн изменениями) или **Login without any change** (подключение без каких-либо изменений): Трассировки продолжают работать.
- **Login with download** (подключение с загрузкой): Трассировки в контроллере удаляются и необходима их новая загрузка.

Онлайн изменения конфигурации графика трассировки

Диалоговое окно **Trace Configuration** с опцией **Record Settings** и диалоговое окно **Trace Configuration** с опцией **Variable Settings** доступны в режиме онлайн, и совсем немного изменений в конфигурации трассировки может быть произведено, пока трассировка работает. Если это невозможно, например, когда изменяется имя сигнала трассировки, трассировка останавливается, и требуется новая загрузка.

Онлайн навигация графика трассировки

Отображаемый диапазон зарегистрированных значений переменных трассировки зависит не только от конфигурации трассировки. Он также может быть переназначен функциями прокрутки и масштабирования, доступными в меню **Trace**, панели инструментов или через использование комбинации клавиш. За информацией о том, как перемещаться в графике трассировки, обращайтесь к главе *Быстрые команды клавиатуры* *(см. стр. 471)*.

Раздел 24.4

Действия клавиатуры для графиков трассировки

Быстрые команды клавиатуры

Обзор

Следующая таблица описывает действия клавиатуры и мыши:

Действия	Действие с клавиатуры	Действие мышью
Прокрутка графика трассировки горизонтально вдоль оси времени.	Нет курсора трассировки: ● Стрелки Влево/Вправо ● На большие расстояния: CTRL + Стрелки Влево/Вправо 1 или 2 курсора трассировки: ● ALT + Стрелки Влево/Вправо ● На большие расстояния: CTRL + ALT +Стрелки Влево/Вправо	Прокрутка графика методом перетаскивания. Это индицируется различным видом курсора мыши.
Прокрутка графика трассировки горизонтально вдоль вертикальной оси.	Стрелки Вверх/Вниз На большие расстояния: CTRL + Стрелки Вверх/Вниз	Используйте CTRL + перетаскивание.
Увеличение до прямоугольника (окна), который выделяется с помощью мыши.	–	Используйте команду **Mouse Zooming** *(см.SoMachine, Menu Commands, Online Help)*.
Сдвиг чёрного курсора трассировки.	Стрелки Влево/Вправо На большие расстояния: CTRL + Стрелки Влево/Вправо	Нажмите чёрный треугольник курсора трассировки, перетаскивайте его вдоль оси X пока Вы не отпустите его.
Сдвиг серого курсора трассировки.	SHIFT + Стрелки Влево/Вправо На большие расстояния: CTRL + SHIFT + Стрелки Влево/Вправо	Нажмите серый треугольник курсора трассировки, перетаскивайте его вдоль оси X, пока Вы его не отпустите.
Сжатие оси времени. В мультиканальном режиме оси времени для всех графиков сжимаются.	–	Используйте колесо мыши. Или используйте команду **Compress** *(см. SoMachine, Menu Commands, Online Help)*.
Растяжение оси времени. В мультиканальном режиме оси времени для всех графиков растягиваются.	+	Используйте колесо мыши. Или используйте команду **Stretch** *(см. SoMachine, Menu Commands, Online Help)*.

Действия	Действие с клавиатуры	Действие мышью
Сжатие оси Y. В мультиканальном режиме ось Y для выбранных диаграмм сжимается.	CTRL + –	CTRL + колесо мыши
Растягивание оси Y. В мультиканальном режиме ось Y для выбранных диаграмм растягивается.	CTRL + +	CTRL + колесо мыши
Выбор следующего нижнего графика в мультиканальном режиме	TAB	Нажмите на невыбранную диаграмму для её выбора.

Глава 25

Редактор конфигурации символов

Содержание этой главы

Эта глава содержит следующие темы:

Тема	Страница
Редактор конфигурации символов	474
Конфигурация символов	477
Добавление конфигурации символов	478

Редактор конфигурации символов

Обзор

Функции конфигурации символов позволяют Вам создавать описания символов. Символы и переменные, которые они представляют, могут затем быть доступны внешним приложениям, таким как Vijeo-Designer или OPC сервер.

Для конфигурирования символов в приложении дважды щёлкните узел **Symbol Configuration** в дереве **Tools tree**. Открывается вид редактора конфигурации символов.

Редактор конфигурации символов

Редактор содержит таблицу. В зависимости от установленного фильтра, она показывает доступные переменные или только те, которые уже выбраны для конфигурации символов. С этой целью причастные POU или библиотеки перечислены в столбце **Symbols**. Вы можете раскрыть их с целью показать конкретные переменные.

Описание элементов

Кнопка **View**, которая находится в панели инструментов вверху таблицы, позволяет Вам устанавливать следующие фильтры, чтобы уменьшить число отображаемых переменных:

Фильтр	Описание
Unconfigured from Project	Отображаются переменные в проекте, которые ещё не добавлены в конфигурацию символов, но доступны для этой цели.
Unconfigured from Libraries	Также переменные из библиотек, которые ещё не добавлены в конфигурацию символов, но доступны для этой цели в проекте.
Symbols exported via attribute	Эта установка эффективна, когда отображаются только уже сконфигурированные переменные (см. два фильтра, описанных выше). Она имеет результатом, что будут перечислены только те переменные, которые уже выбраны по наличию символов {`attribute 'symbol' := 'read'`} внутри своей декларации. Такие символы отображаются серым цветом. Столбец **Attribute** показывает, чьи права доступа установлены в данное время для переменной с помощью прагмы. Обращайтесь к последующему описанию прав доступа.

Для изменения прав доступа для выбранных элементов щёлкните символ в столбце **AccessRights.**Каждый щелчок мышью будет переключать символ по следующим определениям: чтение+запись, только чтение , только запись .

Столбец **Maximal** показывает, какие права максимально могут быть установлены.

Столбец **Comment** показывает какие-либо комментарии, которые были добавлены к декларации переменной.

При свойстве POU **Link Always** нескомпилированный объект может быть реинтерпретирован и загружен в контроллер. Если это свойство установлено во вкладке **Build** диалогового окна **Properties** выбранного объекта, то все переменные, объявленные в этом объекте, будут доступны, даже если сам объект не упоминается в другом коде. В дополнение, Вы можете использовать прагму{attribute linkalways} *(см. стр. 555)*, чтобы сделать некомпилируемые переменные доступными в конфигурации символов.

Переменные, которые сконфигурированы для экспорта, но в данный момент не действительны в приложении – например из-за удаления своих объявлений – будут показаны красным цветом. Это также применяется к причастным POU или имени библиотеки.

Столбец **Type** также показывает псевдонимы типов данных, как например, `MY_INT :INT`. Для переменной, объявленной с типом данных `MY_INT`, тем самым `MY_INT` объявляется следующим образом:
`TYPE MY_INT : INT; END_TYPE`.

Для получения символов для переменной структурного типа данных, как для других переменных, прежде всего активируйте элемент в столбце **Symbols**. Это обычно вызывает то, что для всех членов структуры символы будут экспортированы в файл символов. Это может привести к большому числу записей в файле символов, что не всегда необходимо. Вы также можете выбрать только определённые члены переменных. Вы можете сделать это в диалоговом окне **Symbol Configuration for Data Type**. Щёлкните кнопку **...** в столбце **Members**, чтобы открыть это диалоговое окно. В случае вложенных типов это диалоговое окно вновь предоставит кнопку для открытия другого диалогового окна конфигурации символов типов данных.

Вид редактора автоматически обновляется при выполнении команды построения. Панель инструментов предоставляет кнопку **Build** для быстрого доступа.

Кнопка панели инструментов **Settings** позволяет Вам активировать опцию **Include comments in XML**. Это приведёт к тому, что комментарии, присвоенные переменным, будут также экспортированы в файл символов.

ПРИМЕЧАНИЕ: Не активируйте опцию **Include comments in XML** для проектов с Vijeo-Designer. Иначе Vijeo-Designer не будет действовать верно и может полностью прекратить функционирование.

По умолчанию файл символов создаётся при запуска генерации кода. Этот файл передаётся в устройство при последующей загрузке. Если Вы хотите создать файл без осуществления загрузки, используйте команду **Generate code**, доступную по умолчанию в меню **Build**.

ПРИМЕЧАНИЕ: Переменные списка глобальных переменных (GVL) будут доступны в конфигурации символов, если, по крайней мере, одна из них используется в программном коде.

В случае использования устройства, которое поддерживает отдельный файл приложения для конфигурации символов (обращайтесь также к главе Конфигурации символов *(см. стр. 477)*) кнопка **Download** будет доступной в панели инструментов. Вы можете использовать её для инициации немедленной новой загрузки файла *<applicationname>._Symbols* в случае, если конфигурация символов была изменена в режиме онлайн.

Конфигурация символов

Обзор

Конфигурация символов используется для создания символов, снабжённых определенными правами доступа. Они позволяют внешний доступ к переменным проекта, например, из Vijeo-Designer. Описание символов будет доступно в файле XML (файле символов) в директории проекта. Оно будет загружено в контроллер вместе с приложением.

Символьная информация

Символы, определённые для приложения, экспортируются в XML-файл в директории проекта (файле символов), когда приложение загружается в контроллер. Этот файл именуется согласно следующему синтаксису:

<projectname>.<devicename>.<applicationname>.xml

Пример: *proj_xy.PLC1.application.xml*

ПРИМЕЧАНИЕ: В случае, когда загрузка в контроллер невозможна, Вы можете создать файл конфигурации символов выполнением команды **Generate code**.
Более подробно о загрузке символьной информации в контроллер вместе с приложением. В зависимости от описания устройства она будет включена в приложение или будет сгенерировано отдельное дочернее приложение. Это также будет перечислено с именем *<applicationname>._Symbols* в виде Applications редактора устройств *(см. стр.116)*

Если конфигурация символов была изменена в режиме онлайн, Вы можете перезагрузить её в контроллер нажатием кнопки **Download** в окне редактора *(см. стр.475)*.

Например, относительно максимального числа приложений в контроллере, символьное приложение должно поддерживаться как обычное приложение.

Добавление конфигурации символов

Предпосылки

Переменные, которые будут обмениваться между контроллером и (несколькими) HMI устройствами, используя открытый протокол SoMachine *(см. SoMachine, Введение)* должны быть опубликованы в контроллере с использованием **Symbol configuration**. Затем они будут доступны как переменные SoMachine в Vijeo-Designer.

Определение конфигурации символов

Чтобы получить доступ к функциям конфигурации символов, добавьте объект конфигурации символов к приложению в **Tools tree**, как описано в параграфе *Открытие конфигурации символов*. Это будет автоматически включено в библиотеку IEC VarAccess.library в **Library Manager**.

Вы можете определить переменные, которые экспортируются в символы в редакторе конфигурации символов *(см. стр. 474)* или через прагмы (атрибут символа *(см. стр. 572)*), которые добавляются к объявлению переменных.

ПРИМЕЧАНИЕ: Переменные списка глобальных переменных (GVL) будут доступны в конфигурации символов, если по крайней мере одна из них используется в программном коде.
Другая возможность предоставляется SFCeditor: Вы можете определить неявно созданные флаги элемента в свойствах элемента *(см. стр. 325)* для экспорта в конфигурацию символов.
Имя символа, создаваемого конфигурацией символов, составляется в соответствии со следующим синтаксисом:

<application name>.<POU name>.<variable name>

Примеры:

MyApplication.PLC_PRG.aMyApplication.GVL.a

Для доступа к переменным определяйте имя символа полностью.

Открытие конфигурации символов

Чтобы открыть **Symbol configuration**, проделайте следующее:

Шаг	Действие
1	Выберите узел **Application** в дереве **Tools tree**, щёлкните кнопку зелёного плюса и выберите команду **Add other objects** → **Symbol configuration...**. **Результат:** Будет открыто диалоговое окно **Add Symbol configuration**.
2	В диалоговом окне **Add Symbol configuration** введите в текстовом поле **Name** для конфигурации символов.
3	Щёлкните кнопку **Add**. **Результат:** Узел **Symbol configuration** создаётся под узлом **Application** в **Tools tree**. **Symbol configuration** отображается на правой стороне.

ПРИМЕЧАНИЕ: Может быть создан только один узел конфигурации символов на устройство.
За деталями обмена переменными между контроллером и HMI частью обращайтесь к главе Обмен данных SoMachine контроллер – HMI*(см. стр. 481)*.

Глава 26

Обмен данных SoMachine контроллер – HMI

Содержание этой главы

Эта глава содержит следующие темы:

Тема	Страница
Определение одной переменной в SoMachine	482
Опубликование переменных в контроллерной части	486
Выбор переменных в части HMI	488
Опубликование переменных в части HMI	489
Параметрирование физической среды передачи	491

Определение одной переменной в SoMachine

Обзор

При публикации переменной(ных) в SoMachine они будут автоматически доступны для использования в приложении Vijeo-Designer HMI.

Для обмена переменными по протоколу SoMachine выполните следующие шаги:

- Создайте переменные в контроллерной части.
- Опубликуйте переменные определением их как **Symbols** в контроллерной части. Теперь они доступны в HMI части как переменные SoMachine.
- Сконфигурируйте физическое соединение (автоматически устанавливается SoMachine).

ПРИМЕЧАНИЕ: Последний шаг не нужен для контроллеров XBTGC, потому что они могут коммуницировать со своими собственными переменными контроллера.

Запрет автоматического экспорта символов в Vijeo-Designer

По умолчанию SoMachine автоматически экспортирует те переменные, которые определены как **Symbols**, в приложение Vijeo-Designer HMI.

Когда символы переданы в Vijeo-Designer, обычно нет необходимости производить передачу каждый раз при Вашем вызове Vijeo-Designer. Если Вы позже добавите или измените символы в Вашем приложении SoMachine после первоначальной передачи символов, Вы можете передать символы в Vijeo-Designer вручную позже. Для сокращения времени, когда Вы открываете Vijeo-Designer, Вы можете запретить автоматическую передачу символов следующим образом:

Шаг	Действие
1	Выберите команду **Options...** из меню **Tools**. **Результат:** Диалоговое окно **Options** будет открыто
2	Выберите запись **Vijeo-Designer** из списка на левой стороне.

Шаг	Действие
3	С правой стороны установите флажок **Disable automatic symbol export**.
4	Нажмите **OK** для закрытия диалогового окна.

ПРИМЕЧАНИЕ: Активирование функции **Disable automatic symbol export** запрещает автоматический экспорт переменных SoMachine, определённых как **Symbols**, в Vijeo-Designer. Для того, чтобы произвести эту передачу вручную щёлкните правой кнопкой узел **Symbol configuration** в окне **Devices** и выполните команду **Export Symbols toVijeo-Designer**. Если вы не произведёте эту ручную передачу, Vijeo-Designer может не показывать верных символов, что в свою очередь может привести к ошибкам, обнаруживаемым в проекте.

ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДУСМОТРЕННОЕ ДЕЙСТВИЕ ОБОРУДОВАНИЯ

Выполните команду Export Symbolsto Vijeo-Designer, если Вы активировали Disable automatic symbol export, до того как Вы начнёте работать в Vijeo-Designer.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

Типы переменных для обмена данными SoMachine–HMI

Следующая таблица перечисляет типы переменных для обмена данными SoMachine–HMI:

Переменная в SoMachine	Переменная в Vijeo-Designer	Комментарий
BOOL	BOOL	--
BYTE	Integer	--
WORD	UINT	--
DWORD	UDINT	--
SINT	Integer	--
INT	INT	--
DINT	DINT	--
USINT	Integer	--
UINT	UINT	--
UDINT	UDINT	--
REAL	REAL	--
STRING	STRING	--
WSTRING	STRING	WSTRING поддерживается в Vijeo-Designer как общий тип STRING. Это означает, что Вы можете или обмениваться с HMI только STRINGS, или только WSTRINGS. Смешивание этих двух типов переменных недопустимо. Если Вы используете WSTRINGS, все Ваши строки должны быть WSTRINGs. Укажите драйверу Vijeo-Designer, что все строки должны быть обработаны как UNICODEWSTRINGs следующим образом: Выберите узел **SoMachine Network** или **SoMachine Combo** в дереве **Navigator** Vijeo-Designer и установите параметр **String Encoding** на значение**Unicode**.
Array	–	В Vijeo-Designer Вы только можете обращаться к элементам массива, не ко всему массиву. **Пример:** Ваш массив состоит из элементов SINT и именуется `myValues`. В Vijeo-Designer Вы можете ссылаться на `myValues[0]` или `myValues[5]` и помещать их в переменные HMI контроллера. Массив не должен содержать больше чем 2048 элементов. Если Вы попытаетесь использовать массив с более чем 2048 элементами в Vijeo-Designer, будет выдано сообщение.
DUT	–	В Vijeo-Designer Вы только можете ссылаться на элементы DUT, но не целый DUT. Это поведение аналогично поведению массивов

Неподдерживаемые типы переменных

Следующие типы переменных не поддерживаются в обмене SoMachine–HMI:

- все 64-битовые целые форматы
- LREAL
- Все форматы time и date
- массивы, начинающиеся не с нулевого элемента: Вы не можете импортировать массив, который определён, например, так: `myArray[1..100]`.
- массивы массивов: Вы не можете импортировать массив, элементами которого являются массивы, такой как `ARRAY [0..9] OFARRAY [0..9] OFINT`. Тем не менее, Вы можете использовать многоразмерные массивы, такие как `ARRAY [0..9, 0..9] OFINT`.

ПРИМЕЧАНИЕ: Переменные из структур `PLC_R` библиотеки PLCSystem не могут быть распределены через **Symbol Configuration** с приложением Vijeo-Designer для HMI целевых платформ (включая HMI контроллеры).

ПРИМЕЧАНИЕ: Не распределяйте ссылки на структурированные переменные в редакторе конфигурации символов, так как значения не будут отображаться в HMI.

Для дополнительной информации по типам переменных для обмена данными SoMachine–HMI см. Vijeo-Designer онлайн справку.

Длина идентификатора

В Vijeo-Designer максимальная длина имени символа ограничено 32 символами.

Опубликование переменных в контроллерной части

Обзор

Опубликовывайте переменные в контроллерной части приложения SoMachine в редакторе **Symbol configuration** или в виде **Variables** каталога программного обеспечения *(см. стр. 34)* POU.

Опубликование переменных в редакторе конфигурации символов

Для опубликования переменных в редакторе **Symbolconfiguration**, проделайте следующее:

Шаг	Действие
1	Создайте узел **Symbol Configuration** под узлом **Application** в дереве **Tools tree**, как описано в главе *Добавление конфигурации символов (см. стр. 478)*.
2	Дважды щёлкните узел **Symbol Configuration**, чтобы открыть редактор **Symbol configuration.**
3	В редакторе **Symbol configuration** выберите элементарные переменные, которые Вы желаете опубликовать для коммуникаций с одним или несколькими HMI терминалами с помощью установки или отмены флажков в столбце **Symbols**: Вы также можете назначить права доступа по чтению/записи для каждой переменной индивидуально в столбце **AccessRights**. За дальнейшей информацией обращайтесь к описанию редактора Symbolconfiguration*(см. стр. 474)*. **Примечание:** Только переменные элементарных типов данных доступны для обмена с HMI терминалами.
4	Чтобы Ваши установки стали действительными, щёлкните связь **Download** в редакторе **Symbolconfiguration**

ПРИМЕЧАНИЕ: Механизм публикации занимает около 50 кБайт в контроллере. Каждая опубликованная переменная занимает 11 байт в приложении контроллера.

Опубликование переменных в виде Variables каталога Software

Для опубликования переменных в виде **Variables** каталога программного обеспечения *(см. стр. 34)* проделайте следующее:

Шаг	Действие
1	Откройте вид **Variables** в каталоге программного обеспечения *(см. стр. 34)*.
2	Для публикации переменной установите соответствующий флажок в столбце **Publish.**

ПРИМЕЧАНИЕ: Проверьте, что POU выбранных переменных вызывается в задаче. Иначе выбранные переменные не будут опубликованы.

Выбор переменных в HMI части

Выбор переменных

Те переменные, которые были опубликованы в контроллерной части, непосредственно доступны в HMI части.

В **Expression Editor Pad** Vijeo-Designer выберите вкладку **SoMachine** для прямого доступа к переменным, опубликованным в SoMachine.

Для дальнейшей информации обращайтесь к онлайн справке Vijeo-Designer.

Публикация переменных в HMI части

Поддерживаемые типы переменных

Следующие типы переменных могут быть опубликованы в Vijeo-Designer, чтобы сделать их доступными для всего проекта SoMachine:

- BOOL
- DINT
- INT
- UINT
- UDINT
- Integer
- REAL
- STRING

Процедура

Чтобы опубликовать вышеупомянутые типы переменных, выполните следующее:

Шаг	Действие
1	В Vijeo-Designer **Variable Editor** выберите те переменные, которые Вы хотите опубликовать.
2	Щёлкните правой кнопкой по выбранным переменным и выполните команду **Move to SoMachine** из контекстного меню. **Результат**: Будет отображено диалоговое окно **Move to SoMachine**.
3	В диалоговом окне **Move to SoMachine** откройте поддиректории устройств, определённые в SoMachine, чтобы увидеть уровни, где определены переменные (POU или GVL).

Шаг	Действие
4	Выберите POU или GVL, к которой Вы хотите добавить выбранную(ые) переменную(ные) и щёлкните **OK**. **Результат:** Выбранные переменные перемещаются в выбранные SoMachine POU или GVL и доступны повсюду в проекте SoMachine.

Параметризация физической среды передачи

Обзор

Рабочий обмен данными между контроллером и HMI выполняется по различной среде передачи в зависимости от выбранного аппаратного обеспечения.

Пример конфигурации

Уставки по умолчанию, приведенные ниже, верны для коммуникаций между M238 и панелью HMI по последовательной линии RS485 при использовании кабеля XBTZ9008 (последовательная линия SubD-RJ45).

Конфигурация M238 с HMI панелью:

Конфигурация последовательной линии M238 контроллера

Параметр	Значение
Physical Medium	RS485
Baud rate	115200
Parity	none
Data bits	8
Stop bits	1

Конфигурация IO-Manager панели HMI, при использовании драйвера: SoMachine – сеть с по крайней мере 1 Scan- Group (за дальнейшей информацией обращайтесь к Vijeo-Designer онлайн справке).

Параметр	Значение
Physical Medium	RS485
Baud rate	115200
Parity	none
Data bits	8
Stop bits	1
Equipment Name	Имя устройства контроллера (доступно в диалоге коммуникационных уставок)

Configuration of XBTGC:

Device (Устройство)	Конфигурация
XBTGC controller	не требуется конфигурации
HMI panel	IO-Manager
Driver	SoMachine - Combo с по крайней мере 1 Scan- Group

Часть VII
Справка по программированию

Содержание этой главы

Эта часть содержит следующие главы:

Глава	Название главы	Страница
27	Объявление переменных	495
28	Типы данных	577
29	Инструкции по программированию	603
30	Операторы	617
31	Операнды	709

Глава 27

Объявление переменных

Содержание этой главы

Эта глава содержит следующие разделы:

Раздел	Тема	Страница
27.1	Объявление	496
27.2	Типы переменных	513
27.3	Типы методов	524
27.4	Инструкции прагм	529
27.5	Прагмы атрибутов	542
27.6	Умная функциональность кодирования	575

Раздел 27.1

Объявление

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Общая информация	497
Рекомендации по наименованию идентификаторов	500
Инициализация переменных	505
Объявление	506
Режим «быстрых клавиш»	507
Объявление AT	508
Ключевые слова	509

Общая информация

Обзор

Вы можете декларировать переменные:
- в виде **Variables** Каталога программного обеспечения **Software Catalog** (*см. стр. 33*)
- в редакторе объявления **Declaration Editor** POU (*см. стр. 372*)
- через диалоговое окно **Auto Declare** (*см. стр. 506*)
- в редакторе DUT
- в редакторе GVL

Вид (в табличном редакторе объявления это называют **Scope**) переменных, которые будут объявлены, определен ключевыми словами, охватывающими объявление одной или нескольких переменных. В текстовом редакторе объявления (*см. стр. 372*), общее объявление переменных охвачено `VAR` и `END_VAR`.

Для дальнейших видов объявлений переменной обратитесь к:
- `VAR_INPUT`
- `VAR_OUTPUT`
- `VAR_IN_OUT`
- `VAR_GLOBAL`
- `VAR_TEMP`
- `VAR_STAT`
- `VAR_EXTERNAL`
- `VAR_CONFIG`

Ключевые слова типа переменной могут быть добавлены ключевыми словами атрибута (*см. стр. 518*)

Пример: `RETAIN (VAR_INPUT RETAIN)`

Синтаксис

Синтаксис для объявления переменной:

<Identifier> {`AT`<address>}:<data type> {:=<initialization>};

Части в фигурных скобках {} дополнительные.

Идентификатор

Идентификатор - это имя переменной.

Рассмотрите следующие факты при определении идентификатора.

- не позволены никакие пробелы или специальные символы
- отсутствие чувствительности к регистру: `VAR1, Var1` и `var1` - всё одна и та же переменная
- распознавание символа нижней черты: `A_BCD` и `AB_CD` считают двумя различными идентификаторами. Не используйте больше одного символа нижней черты подряд.
- неограниченная длина
- рекомендации относительно многократного использования (см. следующий параграф),

Кроме того, рассмотрите рекомендации, данные в главе Рекомендации по наименованию идентификаторов (*см. стр. 500*).

Многократное использование идентификаторов (пространства имен)

Следующее очерчивает инструкции относительно многократного использования идентификаторов:

- Не создавайте идентификатор, который идентичен ключевому слову.
- Двойное использование идентификаторов не позволено локально.
- Многократное использование идентификатора позволено глобально: у локальной переменной может быть то же имя что у глобальной. В этом случае у локальной переменной в POU будет приоритет.
- У переменной, определенной в списке глобальных переменных (GVL), может быть то же имя как у переменной, определенной в другом списке глобальной переменной (GVL). В этом контексте считайте следующее расширяющими IEC 61131-3 функциями:
  - Оператор глобальной области видимости: путь экземпляра, запускающийся с точки (.), открывает глобальную область видимости. Так, если есть локальная переменная, например `ivar`, с тем же именем, как глобальная переменная, `.ivar` относится к глобальной переменной.
  - Вы можете использовать имя списка глобальной переменной (GVL) как пространство имен для включенных переменных. Вы можете объявить переменные с тем же именем в различных списках глобальной переменной (GVL). К ним можно получить доступ в частности, предшествуя имени переменной с именем списка.

    Пример

    ```
    globlist1.ivar := globlist2.ivar;
    ```

    (* `ivar` из `globlist2` в библиотеке `lib1` копируется `ivar` в GVL `globlist1` *),
  - К переменным, определенным в списке глобальной переменной включенной библиотеки, можно получить доступ согласно синтаксису <library namespace>.<name of GVL>.<variable>.

    Пример:

    ```
    globlist1.ivar := lib1.globlist1.ivar
    ```

    (* `ivar` из `globlist2` в библиотеке `lib1` скопирован в `ivar` в GVL `globlist1` *)

- Для библиотеки также определено пространство имен, когда оно включено через менеджера библиотеки (**Library Manager)**. Таким образом, Вы можете получить доступ к модулю библиотеки или переменной через <library namespace>.<modulename|variablename>. Полагайте, что в случае вложенных библиотек пространства имен всех заинтересованных библиотек должны быть утверждены последовательно.

  Пример: Если на `Lib1` ссылается `Lib0`, к модулю `fun`, являющемуся частью `Lib1`, получают доступ через `Lib0. Lib1.fun:`

  `ivar: = Lib0. Lib1.fun (4, 5);` (* возвращаемое значение `fun` скопировано в переменную `ivar` в проекте *)

  ПРИМЕЧАНИЕ: Когда флажок **Publish all IEC symbols to that project as if this reference would have been included there directly.** был активирован в диалоговом окне **Properties** библиотеки `Lib`, на которую ссылаются, к модулю `fun` можно также получить доступ непосредственно через `Lib0.fun`.

AT <address>

Вы можете соединить переменную непосредственно с определенным адресом (*см. стр. 508*), используя ключевое слово `AT`.

В функциональных блоках Вы можете также определить переменные с неполными операторами адреса. Чтобы такая переменная могла использоваться в локальном экземпляре, запись должна существовать для нее в переменной конфигурации.

Тип

Допустимый тип данных *(см. стр. 578)*, дополнительно расширенный инициализацией: = :=< initialization>; (*см. стр. 505*),

Инструкции прагм

Дополнительно Вы можете добавить инструкции прагм *(см. стр. 529)* в разделе описаний объекта, чтобы влиять на генерацию кода в различных целях.

Подсказки

Автоматическое объявление (*см. стр. 506*) переменных также возможно.

Для более быстрого ввода объявлений используйте режим «быстрых клавиш» (*см. стр. 507*).

Рекомендации по наименованию идентификаторов

Обзор

Идентификаторы определяются:
- в объявлении (декларации) переменных (имя переменной)
- в объявлении определяемых пользователем типов данных
- при создании POU (функции, функциональные блоки, программы)

В дополнение к общим элементам, которые рассмотрены при определении идентификатора (обращайтесь к главе *Общая информация* об объявлении переменных (см. стр. 497)), рассмотрите следующие рекомендации, чтобы сделать наименование максимально уникальным:
- Имена переменных (*см. стр. 500*),
- Имена переменных в Библиотеках (*см. стр. 502*),
- Определяемые пользователем типы данных (DUTs) в Библиотеках (*см. стр. 503*),
- Функции, Функциональные блоки, Программы (POU), Действия (*см. стр. 503*),
- POU в Библиотеках (*см. стр. 504*),
- Имена визуализаций (*см. стр. 504*),

Имена переменных

Для наименования переменных в приложениях и библиотеках, следуйте за Венгерской нотацией в максимально возможной степени.

Найдите для каждой переменной значимое, краткое описание. Оно используется в качестве базового имени. Используйте прописную букву для каждого слова базового имени. Используйте строчные буквы для остальных (пример: FileSize).

Тип данных	Нижний предел	Верхний предел	Объем информации	Префикс	Комментарий
BOOL	FALSE	TRUE	1 bit	`x`*	–
				`b`	зарезервирован
BYTE	–	–	8 bit	`by`	строка битов, не для арифметических операций
WORD	–	–	16 bit	`w`	строка битов, не для арифметических операций
DWORD	–	–	32 bit	`dw`	строка битов, не для арифметических операций
LWORD	–	–	64 bit	`lw`	не для арифметических операций
SINT	–128	127	8 bit	`si`	–
* намеренно для логических переменных x выбран в качестве префикса, чтобы отличать от БАЙТА и также чтобы приспособить восприятие программиста IEC (см. адресацию `%IX0.0`).					

Тип данных	Нижний предел	Верхний предел	Объем информации	Префикс	Комментарий
USINT	0	255	8 bit	usi	–
INT	–32,768	32,767	16 bit	i	–
UINT	0	65,535	16 bit	ui	–
DINT	–2,147,483,648	2,147,483,647	32 bit	di	–
UDINT	0	4,294,967,295	32 bit	udi	–
LINT	-2^{63}	$2^{63}-1$	64 bit	li	–
ULINT	0	$2^{64}-1$	64 bit	uli	–
REAL	–	–	32 bit	r	–
LREAL	–	–	64 bit	lr	–
STRING	–	–	–	s	–
TIME	–	–	–	tim	–
TIME_OF_DAY	–	–	–	tod	–
DATE_AND_TIME	–	–	–	dt	–
DATE	–	–	–	date	–
ENUM	–	–	16 bit	e	–
POINTER	–	–	–	p	–
ARRAY	–	–	–	a	–
* намеренно для логических переменных x выбран в качестве префикса, чтобы отличать от БАЙТА и также чтобы приспособить восприятие программиста IEC (см. адресацию `%IX0.0`).					

Простое объявление

Примеры для простых объявлений:

```
bySubIndex: BYTE;
sFileName: STRING;
udiCounter: UDINT;
```

Вложенное объявление

Пример для вложенного объявления, где префиксы присоединены друг к другу в порядке объявлений:

```
pabyTelegramData: POINTER TO ARRAY [0..7] OF BYTE;
```

Экземпляры функционального блока и переменные определяемых пользователем типов данных

Экземпляры функционального блока и переменные определяемых пользователем типов данных получают ярлык для функционального блока или имени типа данных как префикс (например: `sdo`).

Пример

```
cansdoReceivedTelegram: CAN_SDOTelegram;
TYPE CAN_SDOTelegram :      (* prefix: sdo *)
STRUCT
  wIndex:WORD;
  bySubIndex:BYTE;
  byLen:BYTE;
  aby: ARRAY [0..3] OF BYTE;
END_STRUCT
END_TYPE
```

Локальные константы

Локальные константы (c) начинаются с префикса `c` и присоединенной нижней черты, сопровождаемых префиксом типа и именем переменной.

Пример

```
VAR CONSTANT
  c_uiSyncID: UINT := 16#80;
END_VAR
```

Глобальные переменные и глобальные константы

Глобальные переменные снабжены префиксом `g_`, и глобальные константы снабжены префиксом `gc_`.

Пример

```
VAR_GLOBAL
  g_iTest: INT;
END_VAR
VAR_GLOBAL CONSTANT
  gc_dwExample: DWORD;
END_VAR
```

Имена переменных в библиотеках

Структура

В основном обращайтесь к вышеупомянутому описанию для имен переменной. Используйте пространство имен библиотеки в качестве префикса при доступе к переменной в коде приложения.

Пример

```
g_iTest: INT; (декларация)
CAN.g_iTest (реализация, вызов в приложении программы)
```

Определяемые пользователем типы данных (DUT - User-Defined Data Types) в библиотеках

Структура

Имя каждого типа данных структуры состоит из короткого выразительного описания структуры (например, `SDOTelegram`).

Пример (в библиотеке с пространством имен `CAL`):

```
TYPE Day :(
MONDAY,
TUESDAY,
WEDNESDAY,
THURSDAY,
FRIDAY,
SATURDAY,
SUNDAY);
```

Объявление:

```
eToday: CAL.Day;
```

Использование в приложении

```
IF eToday = CAL.Day.MONDAY THEN
```

ПРИМЕЧАНИЕ: Считайте использование пространства имен при использовании DUTs или перечислений объявленным в библиотеках.

Функции, функциональные блоки, программы (POU), действия

Имена функций, функциональных блоков и программ снабжены префиксом выразительным кратким именем POU (например, `SendTelegram`). Как с переменными, первая буква слова имени POU должна всегда быть прописной буквой, тогда как другие должны быть строчными буквами. Рекомендуется составлять имя POU глагола и существительного.

Пример

```
FUNCTION_BLOCK SendTelegram (* prefix: canst *)
```

В разделе описаний предоставьте краткое описание POU как комментарий. Далее по вводам и выводам нужно предоставить комментарии. В случае функциональных блоков вставьте связанный префикс для экземпляров установки непосредственно после имени.

Действия

Действия не получают префикс. Только те действия, которые нужно вызвать только внутренне, то есть самим POU, начинаются с `prv_`.

У каждой функции - по причине совместимости с предыдущими версиями программного обеспечения - как предполагается, есть по крайней мере один параметр. Не используйте структуры в качестве возвращаемых значений во внешних функциях.

POU в библиотеках

Структура

Для создания имен методов применяются те же правила, что и относительно действий. Вводите английские комментарии для возможных вводов метода. Добавьте краткое описание метода к его объявлению. Начинайте имена интерфейса с буквы `I`; например, `ICANDevice`.

ПРИМЕЧАНИЕ: Рассмотрите использование пространства имен при использовании POU объявленных в библиотеках.

Имена визуализаций

Постарайтесь не называть визуализацию подобной другому объекту в проекте, потому что это вызвало бы аномалии в случае изменений визуализации.

Инициализация переменных

Значение инициализации по умолчанию

Значение инициализации по умолчанию равно 0 для всех объявлений, но Вы можете добавить определяемые пользователем значения инициализации в объявлении каждого типа переменной и типа данных.

Определяемые пользователем значения инициализации

Определяемая пользователем инициализация вызвается оператором присваивания **: =** и может быть любым допустимым выражением ST. Таким образом, постоянные значения, а также другие переменные или функции могут быть использованы, чтобы определить значение инициализации. Проверьте, что переменная, используемая для инициализации другой переменной, уже сама инициализирована.

Пример допустимых переменных инициализаций:

```
VAR
var1:INT := 12;  * Integer variable with initial value of 12. *
x : INT := 13 + 8;            * Integer value defined an expression wit h literal
values.*
y : INT := x + fun(4);        * Integer value defined by an expression containing
a function call. NOTE: Be sure that any variables used in the variable
initialization have already been defined. *
z : POINTER TO INT := ADR(y); * POINTER is not described by the IEC6113 1-3:
Integer value defined by an address function; NOTE: The pointer will not be
initialized if the declaration is modified online. *
END_VAR
```

Дополнительная информация

Для получения дополнительной информации обратитесь к следующим описаниям:

- инициализация массивов *(см. стр. 592)*,
- инициализация структур (*см. стр. 595*)
- инициализация переменной с типом поддиапазона *(см. стр.599)*

ПРИМЕЧАНИЕ: Переменные списков глобальных переменных (GVL) инициализируются до локальных переменных POU.

Объявление

Типы объявления

Вы можете объявить переменные вручную при помощи текстового или табличного редактора объявления *(см. стр. 372)*, или автоматически, как объясняется в этой главе.

Автоматическое автообъявление

Вы можете определить в диалоговом окне **Options**, категории **Text editor** → **Editing**, что должно открыть диалоговое окно **Auto Declare**, как только ещё не заявленная строка будет введена в реализационную часть редактора и нажата клавиша ENTER. Это диалоговое окно поддерживает объявление переменной *(см. стр. 497)*.

Ручное автообъявление

Чтобы открыть диалоговое окно **Auto Declare** вручную:

- выполните команду **Auto Declare**, которая по умолчанию доступна в меню **Edit** или
- нажмите клавиши SHIFT+F2

Если Вы выбирали уже заявленную переменную прежде, чем открыть диалоговое окно **Auto Declare**, Вы можете отредактировать объявление этой переменной.

Режим «быстрых клавиш»

Обзор

Редактор объявления *(см. стр. 372)* и другие текстовые редакторы, где производятся объявления, поддерживают режим **«быстрых клавиш»**.

Активируйте этот режим при нажатии CTRL+ENTER, когда Вы закончите строку объявления. Это позволяет Вам использовать **«быстрые клавиши»** вместо того, чтобы полностью ввести объявление.

Поддерживаемые «быстрые клавиши»

Поддерживаются следующие **«быстрые клавиши»**:

- Все идентификаторы до последнего идентификатора строки станут идентификаторами объявлений переменных
- Тип объявления определяется последним идентификатором строки.

В этом контексте выполнены следующие замены:

`B` или `BOOL`	Заменено на	BOOL
`I` или `INT`		INT
`R` или `REAL`		REAL
`S` или `string`		STRING

- Если не был установлен никакой тип через эти правила, автоматически задаётся тип BOOL, и последний идентификатор не будет использоваться в качестве типа (см. пример 1).
- Каждая константа, в зависимости от типа объявления, превратится в инициализацию или строку (см. примеры 2 и 3).
- Адрес (как в `%MD12`) расширен в соответствии с ключевым словом `AT` (см. пример 4).
- Текст после точки с запятой (;) становится комментарием (см. пример 4).
- Все другие символы в строке игнорируются (см., например, восклицательный знак в примере 5).

Примеры

Пример №	«Быстрые клавиши»	Получающееся объявление
1	`A`	`A: BOOL;`
2	`A B I 2`	`A, B: INT := 2;`
3	`ST S 2; A string`	`ST:STRING(2); (* A string *)`
4	`X %MD12 R 5 Real Number`	`X AT %MD12: REAL := 5.0;(* Real Number *)`
5	`B !`	`B: BOOL;`

Объявление AT

Обзор

Чтобы соединить переменную проекта с определенным адресом, Вы можете присвоить переменные адресам в виде **I/O Mapping (**ввода-вывода устройства) в конфигурации контроллера (редактор устройства). Альтернативно Вы можете ввести этот адрес непосредственно в объявление переменной.

Синтаксис

<identifier> AT<address> : <data type>;

Допустимый адрес должен следовать за ключевым словом AT. Для получения дополнительной информации обратитесь к описанию *Адреса (см. стр. 725)*. Рассмотрите возможные перекрытия в случае способа адресации байта.

Это объявление позволяет присваивать содержательное имя к адресу. Любые изменения относительно входящего или исходящего сигнала могут быть сделаны только в единственном месте (например, в объявлении).

Рассмотрите следующее при выборе переменной, которая будет присвоена адресу:

- К переменным, требующим ввода, нельзя получить доступ по записи. Компилятор прерывает это обнаружением ошибки.
- AT объявления могут использоваться только с локальными или глобальными переменными. Они не могут использоваться с переменными ввода и вывода POU.
- AT объявления не позволены в списках персистентных переменных.
- Если AT объявления используются со структурой или участниками функционального блока, все экземпляры получат доступ к той же ячейке памяти этой структуры / функционального блока. Это соответствует статическим переменным в классических языках программирования, таких как C.
- Расположение памяти структур также определено к установленному сроку.

Примеры

```
xCounterHeat7 AT %QX0.0: BOOL;
xLightCabinetImpulse AT %IX7.2: BOOL;
xDownload AT %MX2.2: BOOL;
```

Примечание

Если логические переменные присвоены адресам BYTE, WORD или DWORD, они занимают 1 байт с TRUE (ИСТИНОЙ) или FALSE (ЛОЖЬЮ), а не только первый бит после смещения.

Объяснение: собственно булевские переменные – на самом деле занимают при объявлении 8 битов, и поэтому когда они записаны в другие типы переменных, продвигаются все 8 битов.

Ключевые слова

Обзор

Записывайте в редакторах ключевые слова прописными буквами.

Следующие строки зарезервированы как ключевые слова. Они не могут использоваться в качестве идентификаторов для переменных или POU:

- ABS
- ACOS
- ACTION (используемый только в экспортном формате)
- ADD
- ADR
- AND
- ANDN
- ARRAY
- ASIN
- AT
- ATAN
- BITADR
- BOOL
- BY
- BYTE
- CAL
- CALC
- CALCN
- CASE
- CONSTANT
- COS
- DATE
- DINT
- DIV
- DO
- DT
- DWORD
- ELSE
- ELSIF
- END_ACTION (используемый только в экспортном формате)
- END_CASE
- END_FOR
- END_FUNCTION (используемый только в экспортном формате)
- END_FUNCTION_BLOCK (используемый только в экспортном формате)
- END_IF
- END_PROGRAM (используемый только в экспортном формате)
- END_REPEAT
- END_STRUCT

- END_TYPE
- END_VAR
- END_WHILE
- EQ
- EXIT
- EXP
- EXPT
- FALSE
- FOR
- FUNCTION
- FUNCTION_BLOCK
- GE
- GT
- IF
- INDEXOF
- INT
- JMP
- JMPC
- JMPCN
- LD
- LDN
- LE
- LINT
- LN
- LOG
- LREAL
- LT
- LTIME
- LWORD
- MAX
- METHOD
- MIN
- MOD
- MOVE
- MUL
- MUX
- NE
- NOT
- OF
- OR
- ORN
- PARAMS
- PERSISTENT
- POINTER
- PROGRAM

- R
- READ_ONLY
- READ_WRITE
- REAL
- REFERENCE
- REPEAT
- RET
- RETAIN
- RETC
- RETCN
- RETURN
- ROL
- ROR
- S
- SEL
- SHL
- SHR
- SIN
- SINT
- SIZEOF
- SUPER
- SQRT
- ST
- STN
- STRING
- STRUCT
- SUPER
- SUB
- TAN
- THEN
- THIS
- TIME
- TO
- TOD
- TRUE
- TRUNC
- TYPE
- UDINT
- UINT
- ULINT
- UNTIL
- USINT
- VAR
- VAR_ACCESS (используемый только специально, в зависимости от аппаратных средств)
- VAR_CONFIG

- VAR_EXTERNAL
- VAR_GLOBAL
- VAR_IN_OUT
- VAR_INPUT
- VAR_OUTPUT
- VAR_STAT
- VAR_TEMP
- WHILE
- WORD
- WSTRING
- XOR
- XORN

Кроме того, операторы преобразования, перечисленные в **Input Assistant**, также поддерживаются как ключевые слова.

Раздел 27.2

Типы переменных

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Типы переменных	514
Ключевые слова атрибутов для типов переменных	518
Конфигурация переменных – VAR_CONFIG	522

Типы переменных

Обзор

Эта глава предоставляет дополнительную информацию о следующих типах переменных

- VAR локальные переменные *(см. стр. 514)*
- VAR_INPUT входные переменные *(см. стр. 514)*
- VAR_OUTPUT выходные переменные *(см. стр. 515)*
- VAR_IN_OUT входные и выходные переменные *(см. стр. 515)*
- VAR_GLOBAL глобальные переменные *(см. стр. 516)*
- VAR_TEMP временные переменные *(см. стр. 516)*
- VAR_STAT статические переменные *(см. стр. 516)*
- VAR_EXTERNAL внешние переменные *(см. стр. 517)*

Локальные переменные – VAR

Между ключевыми словами VAR и END_VAR, объявляются все локальные переменные POU *(см. стр. 497)*. У них нет внешнего соединения; другими словами, они не могут быть записаны извне.

Рассмотрите возможность добавления атрибута *(см. стр. 518)* к VAR.

Пример

```
VAR
   iLoc1:INT; (* 1. Local Variable*)
END_VAR
```

Входные переменные – VAR_INPUT

Между ключевыми словами VAR_INPUT и END_VAR, объявляются все переменные *(см. стр. 497)*, которые служат входными переменными для POU. Это означает, что в позиции вызова значения переменных могут быть обеспечены вместе с вызовом.

Рассмотрите возможность добавления атрибута *(см. стр. 518)*

Пример

```
VAR_INPUT
   iIn1:INT (* 1. Inputvariable*)
END_VAR
```

Выходные переменные – VAR_OUTPUT

Между ключевыми словами VAR_OUTPUT и END_VAR объявляются все переменные, которые служат выходными переменными POU. Это означает, что эти значения передаются на POU, которое выполняет вызов.

Рассмотрите возможность добавления атрибута *(см. стр. 518)* к VAR_OUTPUT.

Пример

```
VAR_OUTPUT
   iOut1:INT; (* 1. Outputvariable*)
END_VAR
```

Выходные переменные в функциях и методах:

Согласно IEC 61131-3 проекту 2, у функций (и методов) могут быть дополнительные выходы. Вы можете присвоить их в вызове функции как показано в следующем примере.

Пример

```
fun(iIn1 := 1, iIn2 := 2, iOut1 => iLoc1, iOut2 => iLoc2);
```

Переменные входа и выхода – VAR_IN_OUT

Между ключевыми словами VAR_IN_OUT и END_VAR объявляются все переменные *(см. стр. 497)*, которые служат одновременно переменными ввода и вывода для POU.

ПРИМЕЧАНИЕ: С переменными типа IN_OUT, значение переданной переменной изменяется (передаётся как указатель, Вызов по ссылке). Это означает, что входное значение для таких переменных не может быть константой. Поэтому даже переменные VAR_IN_OUT функционального блока не могут быть считаны или записаны непосредственно снаружи через <FBinstance>.<InOutVariable>.

ПРИМЕЧАНИЕ: Не присваивайте символы типа бита (такие как %MXaa.b или переменные BOOL, которые расположены на таком адресе типа бита) параметрам VAR_IN_OUT BOOL-типа функциональных блоков. Если какие-либо такие присвоения обнаружены, о них сообщают как об обнаруженной ошибке сборки (**Build)** в виде сообщений (**Messages**)(см. *SoMachine, Menu Commands, Online Help*).

Пример

```
VAR_IN_OUT
   iInOut1:INT; (* 1. Inputoutputvariable *)
END_VAR
```

Глобальные переменные – VAR_GLOBAL

Вы можете объявить нормальные переменные, константы, внешние, или остаточные переменные, которые известны всюду в проекте, как глобальные переменные. Чтобы объявить глобальные переменные, используйте списки глобальных переменных (GVL). Вы можете добавить GVL, выполнив команду **Add Object** (по умолчанию в меню **Project**).

Объявите переменные локально между ключевыми словами `VAR_GLOBAL` и `END_VAR`.

Рассмотрите возможность добавления атрибута *(см. стр. 518)* к `VAR_GLOBAL`.

Переменная распознаётся как глобальная переменная по предшествующей точке, например, .iGlobVar1. Для получения дальнейшей информации при многократном использовании имен переменной, оператора точки `dot (.)` и пространства имен обращайтесь к главе *Оператор глобального образа (см. стр. 706).*

Глобальные переменные могут быть объявлены только в списках глобальных переменных (GVLs). Они служат для управления глобальными переменными в проекте. Вы можете добавить GVL, выполнив команду **Add Object** (по умолчанию в меню **Project)**.

ПРИМЕЧАНИЕ: у переменной, определенной локально в POU с тем же именем как глобальная переменная, будет приоритет в POU.

ПРИМЕЧАНИЕ: Глобальные переменные инициализированы перед локальными переменными of POU.

Временные переменные – VAR_TEMP

Эта функция - расширение IEC 61131-3 стандарта.

Временные переменные (ре)инициализуются при каждом вызове POU. Объявления VAR_TEMP возможны только в программах и функциональных блоках. Эти переменные также доступны только в теле программы POU или функционального блока.

Объявите переменные локально между ключевыми словами `VAR_TEMP` и `END_VAR`.

ПРИМЕЧАНИЕ: Вы можете использовать `VAR_TEMP` вместо `VAR`, чтобы уменьшить пространство памяти, необходимое POU (например, в функциональном блоке, если переменная используется только временно).

Статические переменные – VAR_STAT

Эта функция – расширение IEC 61131-3 стандарта

Статические переменные могут использоваться в функциональных блоках, методах и функциях. Объявите их локально между ключевыми словами `VAR_STAT` и `END_VAR`. Они инициализируются при первом вызове соответствующей POU.

Так же как глобальные переменные, статические переменные не теряют свое значение после выхода из POU, в котором они объявлены. Они совместно используются POU, в которых они объявлены (например, несколько экземпляров функционального блока, функций или методов совместно используют одну и ту же статическую переменную). Они могут использоваться, например, в функции счетчика для количества вызовов функции.

Рассмотрите возможность добавления атрибута *(см. стр. 518)* к `VAR_STAT`.

Внешние переменные – VAR_EXTERNAL

Это глобальные переменные, которые импортированы в POU.

Объявите их локально между ключевыми словами VAR_EXTERNAL и END_VAR и в списке глобальных переменных (GVL). Объявление и глобальное объявление должны быть идентичными. Если глобальная переменная не будет существовать, будет выведено сообщение.

ПРИМЕЧАНИЕ: Нет необходимости определять переменные как внешние. Эти ключевые слова обеспечены, чтобы поддержать совместимость с IEC 61131-3.

Пример

```
VAR_EXTERNAL
   iVarExt1:INT; (* 1st external variable *)
END_VAR
```

Ключевые слова атрибутов для типов переменных

Обзор

Вы можете добавить следующие ключевые слова атрибутов к объявлению *(см. стр. 497)* типа переменной, чтобы определить их вид:

- `RETAIN`: обращайтесь к *Retain Variables (см. стр. 518)*
- `PERSISTENT`: обращайтесь к *Persistent Variables (см. стр. 519)*
- `CONSTANT`: обращайтесь к *Константам – CONSTANT (см. стр. 520)*, *Typed Literals (см. стр. 521)*

Сохраняемые переменные – `RETAIN, PERSISTENT`

Остаточные (реманентные) переменные могут сохранить свое значение всюду во время обычного периода выполнения программы. Объявите их, как retain (сохраняемые) переменные или, еще более строго, как persistent (персистентные) переменные.

Декларация определяет степень сопротивления остаточной переменной в случае перезапусков, загрузок или перезагрузки контроллера. В приложениях, главным образом, используется комбинация обоих реманентных флагов (обратитесь к *Персистентным переменным (см. стр. 519)*).

ПРИМЕЧАНИЕ: Объявление `VAR PERSISTENT` интерпретируется таким же образом как `VAR PERSISTENT RETAIN` или `VAR RETAIN PERSISTENT`.

ПРИМЕЧАНИЕ: Использование команды *(см. SoMachine, Menu Commands, Online Help)* **Add all instance paths** добавляет переменные, объявленные в качестве персистентных в объект **Persistent list**.

Сохранение переменных

Переменные, объявленные как сохраняемые переменные, сохраняются в области энергонезависимой памяти. Чтобы объявить этот вид переменной, используют ключевое слово RETAIN в разделе описаний POU или в списке глобальной переменной.

Пример

```
VAR RETAIN
   iRem1 : INT; (* 1. Retain variable*)
END_VAR
```

Использование интерфейсов или функциональных блоков из библиотек System Configuration вызовет системные исключения, которые могут сделать контроллер неоперабельным, требуя перезапуска.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННАЯ РАБОТА ОБОРУДОВАНИЯ

- Не используйте интерфейсы из библиотеки SystemConfigurationItf в разделе сохранения программы (`VAR_RETAIN`).
- Не используйте функциональные блоки из библиотеки SystemConfiguration в разделе сохранения программы (`VAR_RETAIN`).

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

Сохраняемые переменные поддерживают своё значение даже после непредвиденного завершения работы контроллера, а также после нормального цикла выключения и включения питания контроллера (или при выполнении **Online** команды **Reset Warm**). При перезапуске программы сохраненные значения будут далее обработаны. Другие (несохраняемые) переменные вновь инициализируются, или их значениями инициализации или их значениями инициализации по умолчанию (в случае, если никакое значение инициализации не было объявлено).

Например, Вы можете захотеть использовать сохраненное значение при работе как часть в расчёте производительности машины, которая должна продолжаться после перебоя в питании.

Сохраняемые (retain) переменные, однако, повторно инициализируются при выполнении **Online** команды **Reset origin** и, в отличие от персистентных переменных, при выполнении **Online** команды **Reset cold** или в ходе загрузки приложения.

ПРИМЕЧАНИЕ: Только специальные переменные, определенные как `VAR RETAIN`, сохраняются в энергонезависимой памяти. Однако локальные переменные, определенные как `VAR RETAIN` в функциях, не сохраняются в энергонезависимой памяти. Определение `VAR RETAIN` локально в функциях, недействительно.

Персистентные переменные

Персистентные переменные идентифицированы ключевым словом `PERSISTENT (VAR_GLOBAL PERSISTENT)`. Они повторно инициализируются только при выполнении **Online** команды **Reset origin**. В отличие от сохраняемых переменных, они поддерживают свои значения после загрузки.

ПРИМЕЧАНИЕ: Не используйте в объявлении `AT` в сочетании с `VAR PERSISTENT`.

Пример применения:

Счетчик часов работы, который должен продолжать расчёт даже после перебоя в питании или загрузки. Обратитесь к конспективной таблице поведения сохраняемых переменных *(см. стр. 520)*.

Вы можете объявить персистентные переменные только в специальном списке глобальных переменных объектов типа персистентных переменных, который присвоен приложению. Вы можете добавить только один такой список к приложению.

ПРИМЕЧАНИЕ: Объявление `VAR_GLOBAL PERSISTENT` имеет тот же эффект, как объявление `VAR_GLOBAL PERSISTENT RETAIN` или `VAR_GLOBAL RETAIN PERSISTENT`.

Как и сохраняемые переменные, персистентные переменные сохраняются в отдельной области памяти.

Пример

```
VAR_GLOBAL PERSISTENT RETAIN
  iVarPers1 : DINT;         (* 1. Persistent+Retain Variable App1 *)
  bVarPers : BOOL;          (* 2. Persistent+Retain Variable App1 *)
END_VAR
```

ПРИМЕЧАНИЕ: Персистентные переменные могут быть объявлены только в объекте **Persistent list**. Если они будут объявлены в другом месте, они будут вести себя как сохраняемые переменные, и о них будет сообщено как об обнаруженной ошибке **Build** в виде сообщений (**Messages)**. (Сохраняемые переменные могут быть объявлены в списках глобальных переменных или в POU.)

При каждой перезагрузке приложения список персистентных переменныхй на контроллере будет проверен с тем же из проекта. Список на контроллере идентифицируется именем приложения. В случае несоответствий Вам предложат повторно инициализировать все персистентные переменные (см. *SoMachine, Menu Commands, Online Help*) приложения. Несоответствие может следовать из переименования или удаления или других модификаций существующих объявлений в списке.

ПРИМЕЧАНИЕ: Тщательно рассмотрите любые модификации в разделе описаний списка персистентных переменных и эффекте результатов относительно реинициализации.

Вы можете добавить новые объявления только в конце списка. Во время загрузки они обнаружены как новые и не потребуют реинициализации полного списка. Если Вы изменяете имя или тип данных переменной, это будет обработано как новое объявление и вызовет реинициализацию переменной в следующем онлайновом изменении или загрузке.

Поведение сохраняемых переменных

Консультируйтесь с Руководством по программированию (*Programming Guide)*, определенным для Вашего контроллера, для получения дополнительной информации о поведении сохраняемых переменных.

Константы – CONSTANT

Константы идентифицированы ключевым словом CONSTANT. Вы можете объявить их локально или глобально.

Синтаксис

```
VAR CONSTANT<identifier>:<type> := <initialization>;END_VAR
```

Пример

```
VAR CONSTANT
  c_iCon1:INT:=12; (* 1. Constant*)
END_VAR
```

Обратитесь к главе *Операнды* *(см. стр. 709)* за списком возможных констант.

Введенные литералы

Обычно, в использовании IEC констант, будет использоваться самый маленький возможный тип данных. Если должен использоваться другой тип данных, это может быть достигнуто с помощью введенных литералов без необходимости явного объявления констант. Для этого константе предоставят префикс, который определяет тип.

Синтаксис

<type>#<literal>;

<type>	определяет желаемый тип данных возможные записи: BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, LREAL Пишите тип прописными буквами.
<literal>	определяет константу Введите данные, которые соответствуют типу данных, определенном под <type>.

Пример

```
iVar1:=DINT#34;
```

Если константа не может быть преобразована в целевой тип без потери данных, выдается сообщение. Вы можете использовать введенные литералы везде, где могут использоваться нормальные константы.

Постоянные в режиме онлайн

Пока настройки по умолчанию **Replace constants** (**File** → **Project Settings** → **Compile options**) активированы, константы в режиме онлайн имеют символ C, предшествующий значению в столбце **Value** в объявлении или в виде просмотра. В этом случае к ним нельзя получить доступ, например, форсированием или записью.

Конфигурация переменных – VAR_CONFIG

Обзор

Вы можете использовать переменную конфигурации, чтобы отобразить переменные функционального блока на изображении процесса, которое находится на устройстве Вх/Вых. Это позволяет избежать потребности указания определенного адреса уже в объявлении переменной функционального блока. Присвоение определенного адреса *(см. стр. 725)* в этом случае сделано централизованно для всех экземпляров функционального блока в глобальном списке VAR_CONFIG.

С этой целью Вы можете присвоить неполные адреса переменным функционального блока, объявленным между ключевыми словами VAR и END_VAR. Используйте звездочку, чтобы идентифицировать эти адреса.

Синтаксис идентификатора

<identifier> AT %<I|Q>* : <data type>

Пример использования не полностью определенных адресов:

```
FUNCTION_BLOCK locio VAR
xLocIn AT %I*: BOOL := TRUE;
xLocOut AT %Q*: BOOL; END_VAR
```

В этом примере определены 2 локальных переменных ввода-вывода: локальный вход (%I*) и локальная выходная переменная (%Q*).

Определите адреса в переменной конфигурации в списке глобальных переменных (GVL) следующим образом:

Шаг	Действие
1	Выполните команду **Add Object**.
2	Добавьте объект **Global Variable List** (GVL) к дереву устройств (**Devices Tree)**.
3	Введите объявления экземпляров переменных с определенными адресами между ключевыми словами VAR_CONFIG и END_VAR.

При определении адресов отметьте следующее:

- Определите экземпляры переменных полным путём экземпляра и разделите индивидуальные POU и имена экземпляров от друг друга по периодам.
- В объявлении введите адрес, класс которого (ввод/вывод) соответствует классу неполного указанного адреса (%I*, %Q*) в функциональном блоке.
- Проверьте, что тип данных согласуется с объявлением в функциональном блоке.

Синтаксис пути экземпляра переменной

<instance variable path> `AT` %<I|Q><location> : <data type>;

Переменные конфигурации, путь экземпляра которых - неправильный, потому что экземпляр не существует, обозначены как обнаруженные ошибки. Ошибка также обнаруживается, если никакая определенная конфигурация адреса не существует для экземпляра переменной, присвоенной неполному адресу.

Пример для переменной конфигурации

Предположите, что следующее определение для функционального блока `locio` – см. предыдущий пример - дано в программе:

```
PROGRAM PLC_PRG VAR
locioVar1: locio;
locioVar2: locio;
END_VAR
```

Тогда исправленная переменная конфигурации будет:

```
VAR_CONFIG
PLC_PRG.locioVar1.xLocIn AT %IX1.0 : BOOL;
PLC_PRG.locioVar1.xLocOut AT %QX0.0 : BOOL;
PLC_PRG.locioVar2.xLocIn AT %IX1.0 : BOOL;
PLC_PRG.locioVar2.xLocOut AT %QX0.3 : BOOL;
END_VAR
```

ПРИМЕЧАНИЕ: Изменения на непосредственно отображенных Вх/Вых сразу показаны в изображении процесса, тогда как изменения в переменных, отображенных через `VAR_CONFIG`, не показаны до конца соответственной задачи.

Раздел 27.3

Типы методов

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Методы FB_init, FB_reinit	525
Метод FB_exit	528

Методы FB_init, FB_reinit

FB_Init

Метод FB_init служит, чтобы повторно инициализировать функциональный блок или структуру. FB_init может быть объявлен явно для функциональных блоков, но в любом случае всегда доступен неявно.

Метод FB_init содержит код инициализации для функционального блока или структуры, как объявлено в разделе описаний соответствующего объекта. Если init метод будет объявлен явно, неявный код инициализации будет введен в этот метод. Программист может добавить дальнейший код инициализации.

ПРИМЕЧАНИЕ: Когда выполнение достигает определяемого пользователем кода инициализации, функциональный блок, соответственно структура, уже полностью инициализирован через неявный код инициализации.

Метод init вызывается после загрузки для каждого заявленного экземпляра функционального блока и каждой переменной типа структуры.

ПРИМЕЧАНИЕ: Выполнение команды **Online Change** перезапишет значения инициализации предыдущими значениями.

Для получения информации о последовательности вызова в случае наследования обратитесь к соответствующему абзацу в главе *Метод FB_exit (см. стр. 528)*.

Для получения информации о возможности определения метода экземпляра функционального блока, который вызовут автоматически после инициализации через FB_init, обратитесь к главе *Атрибут call_after_init (см. стр. 546).*

Интерфейс метода init

```
METHOD fb_init : BOOL VAR_INPUT
bInitRetains : BOOL; // if TRUE, the retain variables are initialized
(warm start / cold start)
bInCopyCode : BOOL; // if TRUE, the instance afterwards
gets moved into the copy code (online change)
END_VAR
```

Возвращаемое значение несущественно.

ПРИМЕЧАНИЕ: Рассмотрите также возможное применение метода выхода и получающийся порядок выполнения (обращайтесь к главе *Метод FB_exit (см. стр. 528)*).

Определяемый пользователем ввод

В методе init Вы можете объявить дополнительные входы функционального блока. Присвойте их в объявлении экземпляра функционального блока.

Пример для метода `init` для функционального блока `serialdevice`:

```
METHOD PUBLIC FB_init : bool
VAR_INPUT
bInitRetains : BOOL; // Initialization of the retain variables
bInCopyCode : BOOL; // Instance moved into copy code
nCOMnum : INT; // Input: number of the COM interface to listen at
END_VAR
```

Объявление функционального блока `serialdevice`:

```
com1: serialdevice (nCOMnum:=1);
com0: serialdevice (nCOMnum:=0);
```

Пример использования `FB_init` для структуры `DUTxy`:

Структура `DUTxy`

```
TYPE DUTxy :
STRUCT
a: INT := 10;
b: INT := 11;
c: INT := 12;
END_STRUCT
END_TYPE
```

Вызов `fb_init` для реализации:

```
PROGRAM PLC_PRG
VAR
dutTest: DUTxy;
xInit: BOOL:=FALSE;
END_VAR
IF xinit THEN // if xinit is set TRUE, then the
reinitialization via fb_init to the values as defined in DUTxy will be done
dutTest.FB_Init(TRUE,TRUE);
ELSE
dutTest.a := 1;
dutTest.b := 2;
dutTest.c := 3;
END_IF
```

FB_reinit

Если метод под названием `FB_reinit` будет объявлен для экземпляра функционального блока, этот метод вызовут, когда экземпляр будет скопирован. Дело обстоит так в онлайновом изменении после модификации объявления функционального блока. Метод вызовет реинициализацию нового экземпляра модуля, который был создан кодом копии. Реинициализация может быть желаема, потому что данные исходного экземпляра будут записаны в новый экземпляр после копирования, даже при том, что Вы хотите получить исходные значения инициализации. У метода `FB_reinit` нет входов. Полагайте, что в отличие от `FB_init` метод должен быть объявлен явно. Если желаема реинициализация реализации основного функцонального блока, `FB_reinit` нужно вызвать явно для этой POU.

У метода `FB_reinit` нет входов.

Для получения информации о последовательности вызова в случае наследования обратитесь к соответствующему абзацу в главе *Метод FB_exit* *(см. стр. 528)*.

Метод FB_exit

Обзор

Метод FB_exit - специальный метод для функционального блока. Он должен быть объявлено явно, поскольку нет никакого неявного объявления. Метод выхода, если присутствует, вызывается для всех заявленных экземпляров функционального блока перед новой загрузкой, в сбросе или во время онлайнового изменения для всех перемещенных или удаленных экземпляров.

Интерфейс метода FB_exit

Есть только один обязательный параметр:

```
METHOD fb_exit : BOOL VAR_INPUT
bInCopyCode : BOOL; // if TRUE, the exit method is called
for exiting an instance that is copied afterwards (online change).
END_VAR
```

Рассмотрите также *метод FB_init* *(см. стр. 525)* и следующий порядок выполнения:

1	метод exit: выход из старого экземпляра	old_inst.fb_exit(bInCopyCode := TRUE);
2	метод init: инициализация нового экземпляра	new_inst.fb_init(bInitRetains := FALSE, bInCopyCode := TRUE);
3	копирование значений функционального блока (код копии)	copy_fub(&old_inst, &new_inst);

Наследование

Помимо этого, в случае наследования, следующая последовательность вызова - TRUE (принятие или POU, используемые для этого перечисления: SubFB EXTENDS MainFB и SubSubFB EXTENDS SubFB):

```
fbSubSubFb.FB_Exit(...);
fbSubFb.FB_Exit(...);
fbMainFb.FB_Exit(...);
fbMainFb.FB_Init(...);
fbSubFb.FB_Init(...);
fbSubSubFb.FB_Init(...);
```

Для FB_reinit применяется следующее:

```
fbMainFb.FB_reinit(...);
fbSubFb.FB_reinit(...);
fbSubSubFb.FB_Init(...);
```

Раздел 27.4

Инструкции прагм

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Инструкции прагм	530
Прагмы сообщений	532
Условные прагмы	534

Инструкции прагм

Обзор

Инструкция прагмы используется, чтобы влиять на свойства одной или нескольких переменных относительно процесса компиляции или предварительной компиляции (препроцессор). Это означает, что прагма влияет на генерацию кода.

ПРИМЕЧАНИЕ: Полагайте, что доступные прагмы не 1:1 реализация директив препроцессору C. Они поддерживаются как нормальные выражения и поэтому могут использоваться только в позициях оператора.

Прагма может определить, будет ли переменная инициализироваться, контролироваться, добавляться к списку параметров, добавляться к списку символов *(см. стр. 572)*, или делать невидимой в менеджере библиотеки (**Library Manager)**. Это может вызвать выходы сообщений во время процесса сборки. Вы можете использовать условные прагмы, чтобы определить, как переменную нужно рассматривать в зависимости от определенных условий. Вы можете также ввести эти прагмы как определения в свойствах компиляции определенного объекта.

Вы можете использовать прагму в отдельной строке или с дополнительным текстом в строке реализации или объявления редактора. В редакторе FBD/LD/IL выполните команду **Insert Label** и замените по умолчанию текстовую Label: в возникающем текстовом поле прагмой. В случае, если Вы хотите установить метку, а также прагму, вставьте сначала прагму и впоследствии метку.

Инструкция прагмы заключена в фигурные скобки

Синтаксис

{ <instruction text> }

Открывающая скобка может быть сразу после имени переменной. Открывающая и закрывающая скобки должно быть в одной и той же строке.

Дополнительная информация

В зависимости от типа и содержания прагмы, прагма воздействует на последующее выражение, соответственно все последующие выражения, пока не встретится одно из следующих условий:

- Оно закончено надлежащей прагмой.
- Та же прагма выполняется с отличными параметрами.
- Достигнут конец кода.

Термин код в этом контексте относится к разделу описаний, части реализации, списку глобальных переменных или описанию типа.

ПРИМЕЧАНИЕ: Инструкции прагм чувствительны к регистру.

Если компилятор не может обоснованно интерпретировать текст инструкции, вся прагма обрабатывается как комментарий и пропускается.

Обращайтесь к следующим типам прагм:
- *ПрагмыMessage (см. стр. 532)*
- *Атрибут Obsolete (см. стр.568)*
- *Атрибут Pragmas (см. стр. 543)*
- *Условные прагмы (см. стр. 534)*
- *Атрибут Symbol (см. стр. 572)*

Прагмы сообщений

Обзор

Вы можете использовать прагмы сообщения, чтобы вызвать выход сообщений в виде **Messages** (по умолчанию в меню **Edit**) во время компиляции (сборки) проекта.

Вы можете вставить инструкцию прагмы в существующую строку или в отдельную строку в текстовом редакторе POU. Прагмы сообщения, расположенные во в настоящее время не определенных секциях реализации кода, не будет рассмотрены при скомпиляции проекта. Для получения дополнительной информации обратитесь к примеру, в котором представлено описание определенного (идентификатор), в главе *Условные прагмы* *(см. стр. 534)*.

Типы прагм сообщений

Есть 4 типа прагм сообщений:

Прагма	Иконка	Тип сообщения
{text 'text string'}	–	тип текста Указанная текстовая строка будет выведена на экран.
{info 'text string'}		информация Указанная текстовая строка будет выведена на экран.
{warning digit 'text string'}		Указанная текстовая строка будет выведена на экран. В отличие от глобальной устаревшей прагмы *(см. стр. 568)*, это предупреждение явно определено для локальной позиции.
{error 'text string'}		тип ошибки Указанная текстовая строка будет выведена на экран.

ПРИМЕЧАНИЕ: Для сообщений информации о типах, предупреждениях и обнаруженной ошибке, Вы можете достигнуть исходной позиции сообщения - это то, куда прагма помещена в POU - выполнив команду **Next Message**. Это не возможно для типа текста.

Пример объявления и реализации в редакторе ST

```
VAR
ivar : INT; {info 'TODO: should get another name'}
bvar : BOOL;
arrTest : ARRAY [0..10] OF INT;
i:INT;
END_VAR
arrTest[i] := arrTest[i]+1;
ivar:=ivar+1;
{warning 'This is an alert'}
{text 'Part xy has been compiled completely'}
```

Выход в виде **Messages**:

Messages

Build

Description	Project	Object	Position
------ Build started: Application: Res.App2 -------			
typify code...			
Compile time before typification: 0 ms			
Compile time after typification: 15 ms			
TODO: should get another name!	TS pragma	NewPOU	Line 3 (Decl)
This is an alert	TS pragma	NewPOU	Line 7 (Impl)
Part xy has been compiled completely	TS pragma	NewPOU	Line 10 (Impl)
Compile complete -- 1 errors, 2 warnings			

Условные прагмы

Обзор

Язык ExST (Расширенный ST) поддерживает несколько условных *Инструкций прагм (см. стр. 530)*, которые влияют на генерацию кода в процессах предварительной компиляции или компиляции.

Код реализации, который будет расценен при компиляции, может зависеть от следующих условий:

- Определенный тип данных или переменная объявлены?
- У типа или переменной есть определенный атрибут?
- У переменной есть определенный тип данных?
- Определенная POU или задача доступны или она является частью дерева вызова, и т.д...

ПРИМЕЧАНИЕ: Невозможно для POU или GVL, объявленных в **POUs Tree**, использовать {define...}, объявленное в приложении. Определения в приложениях будут влиять только на интерфейсы, вставленные ниже соответствующего приложения.

`{define identifier string}`	Во время предварительной обработки все последующие экземпляры идентификатора будут заменены данной последовательностью маркеров, если символическая последовательность не будет пуста (которая позволена и четко определена). Идентификатор остается определенным и в объеме до конца объекта или пока это не определено в директиве {`undefine`}. Используется для условной компиляции *(см. стр. 534)*.
`{undefine identifier}`	Определение препроцессора идентификатора (через `{define}`, см. первую строку этой таблицы) будет удалено, и идентификатор, следовательно, не определен. Если указанный идентификатор не будет в настоящее время определен, эта прагма будет проигнорирована.

`{IF expr}`
...
`{ELSIF expr}`
...
`{ELSE}`
...
`{END IF}`
	Это прагмы для условной компиляции. Требуется, чтобы указанные выражения `exprs` были постоянными во время компиляции; они оцениваются в порядке появления, пока одно из выражений не получит ненулевое значение. Текст, связанный с успешной директивой, предварительно обрабатывается и обычно компилируется; другие игнорируются. Порядок разделов определенный; однако, секции `elsif` и `else` дополнительные, и разделы `elsif` могут появляться произвольно более часто.
	В постоянном `expr` Вы можете использовать несколько операторов условной компиляции (*см. стр. 534*),

Операторы условной компиляции

В рамках постоянного выражения expr условной компиляции pragma (`{if}` или `{elsif}`) (см. предыдущую таблицу), Вы можете использовать несколько операторов. Эти операторы могут не быть не определены или переопределены через `{undefine}` или `{define}`, соответственно. Имейте ввиду, что Вы можете также использовать эти выражения, а также определение, законченное `{define}` в **Compiler defines**: текстовом поле в диалоговом окне свойств объекта **Properties** (**View** → **Properties** → **Build**).

Поддержаны следующие операторы:

- defined (identifier) – определенный (идентификатор) *(см. стр. 535)*
- defined (variable:variable) – определенный (переменная:переменная) *(см. стр. 536)*
- defined (type:identifier) – определенный (тип:идентификатор) *(см. стр. 536)*
- defined (pou:pou-name) – определенный (pou: имя pou) *(см. стр. 536)*
- hasattribute (pou: pou-name, attribute) – имеет признак (pou: имя pou, признак) *(см. стр. 537)*,
- hasattribute (variable: variable, attribute) – имеет признак (переменная: переменная, признак) *(см. стр. 538)*
- hastype (variable:variable, type-spec) – имеет тип (переменная: переменная, специф.тип) *(см. стр. 538)*
- has value (define-ident, char-string) – имеет значение (определ-идентификатор, символьная строка), *(см. стр. 540)*
- NOT operator) – оператор НЕ *(см. стр. 540)*
- operator AND operator) – оператор И оператор *(см. стр. 540)*
- operator OR operator) – оператор ИЛИ оператор *(см. стр. 541)*
- operator *(see page 541)*) – оператор *(см. стр. 541)*

`defined (identifier)` определенный (идентификатор)

Этот оператор затрагивает то, что выражение получает значение TRUE (ИСТИНА), как только идентификатор `identifier` был определен инструкцией `{define}` и позже не был неопределен инструкцией `{undefine}`. Иначе его значение FALSE (ЛОЖЬ).

Пример на `defined (identifier)`:

Предварительное условие: есть 2 приложения `App1` и `App2`. Переменная `pdef1` объявлена в `App2`, но не в `App1`.

```
{IF defined pdef1)}
(* this code is processed in App1 *)
{info 'pdef1 defined'} hugo := hugo + SINT#1;
{ELSE}
(* the following code is only processed in application App2 *)
{info 'pdef1 not defined'} hugo := hugo - SINT#1;
```

Кроме того, пример для сообщения pragma *(см. стр. 532)* включает:

Только информация `pdef1 defined` будет показана в виде сообщений (**Messages)**, когда применение будет собрано, потому что `pdef1` определено. Сообщение **pdef1 not defined** будет показано, когда `pdef1` не будет определено.

defined (variable:variable) - определенный (переменная: переменная)

Когда применяется к переменной, ее значение TRUE (ИСТИНА), если эта конкретная переменная объявлена в текущем объеме. Иначе она FALSE (ЛОЖЬ).

Пример на `defined (variable:variable)`:

Предварительное условие: есть 2 приложения `App1` и `App2`. Переменная `g_bTest` объявлена в `App2`, но не в `App1`.

```
{IF defined (variable:g_bTest)}
(* the following code is only processed in application App2 *)
g bTest := x > 300;
{END_IF}
```

defined (type:identifier) - определенный (тип:идентификатор)

Когда применяется к идентификатору типа, его значение TRUE (ВЕРНО), если тип с конкретным именем объявлен. Иначе она FALSE (ЛОЖЬ).

Пример на `defined (type:identifier)`:

Предварительное условие: есть 2 применения: `App1` и `App2`. Тип данных DUT определен в `App2`, но не в `App1`.

```
{IF defined (type:DUT)}
(* the following code is only processed in application App1 *)
bDutDefined := TRUE;
{END_IF}
```

defined (pou:pou-name) - определенный (pou: имя pou)

Когда применяется к имени POU, её значение TRUE (ИСТИНА), если POU или действие с этим конкретным именем POU определены. Иначе она FALSE (ЛОЖЬ).

Пример на `defined (pou: pou-name)`:

Предварительное условие: Есть 2 приложения `App1` и `App2`. POU `CheckBounds` доступна в `App2`, но не в `App1`.

```
{IF defined (pou:CheckBounds)}
(* the following code is only processed in application App1 *)
arrTest[CheckBounds(0,i,10)] := arrTest[CheckBounds(0,i,10)] + 1;
{ELSE}
(* the following code is only processed in application App2 *)
arrTest[i] := arrTest[i]+1;
{END_IF}
```

`hasattribute (pou: pou-name, attribute)` - имеет признак (pou: имя pou, признак)

Когда применяется к POU, её значение TRUE (ИСТИНА), если этот конкретный признак определен в первой линии декларационной части POU.

Пример на `hasattribute (pou: pou-name, attribute)`:

Предварительное условие: Есть 2 приложения: `App1` и `App2`. Функция `fun1` определена в `App1` и `App2`, но в `App1` имеется видимость признака:

Определение `fun1` в `App1`:

```
{attribute 'vision'}
FUNCTION fun1 : INT
VAR_INPUT
i : INT;
END_VAR
VAR
END_VAR
```

Определение `fun1` в `App2`:

```
FUNCTION fun1 : INT
VAR_INPUT
i : INT;
END_VAR
VAR
END_VAR
```

Инструкция прагмы

```
{IF hasattribute (pou: fun1, 'vision')}
(* the following code is only processed in application App1 *)
ergvar := fun1 ivar);
{END_IF}
```

hasattribute (variable: variable, attribute)

Когда применяется к переменной `variable`, ее значение TRUE (ИСТИНА), если этот конкретный признак определен через инструкцию `{attribute}` в линии перед декларированием переменной.

Пример на `hasattribute (variable: variable, attribute)`:

Предварительное условие: есть 2 приложения `App1` и `App2`. Переменная `g_globalInt` используется в `App1` и `App2`, но в `App1` есть признак `DoCount`:

Объявление `g_globalInt` в `App1`

```
VAR_GLOBAL
{attribute 'DoCount'}
g_globalInt : INT;
g_multiType : STRING;
END_VAR
```

Объявление `g_globalInt` в `App2`

```
VAR_GLOBAL
g_globalInt : INT;
g_multiType : STRING;
END_VAR
```

Инструкция прагмы

```
{IF hasattribute (variable: g_globalInt, 'DoCount')}
(* the following code line will only be processed in App1, because there variable
g_globalInt has got the attribute 'DoCount' *)
g_globalInt := g_globalInt + 1;
{END_IF}
```

hastype (variable:variable, type-spec)

Когда применяется к переменной, ее значение TRUE (ИСТИНА), если у этой конкретной переменной есть указанный специфический тип `type-spec`. Иначе она FALSE (ЛОЖЬ).

Доступные типы данных `type-spec`

- ANY
- ANY_DERIVED
- ANY_ELEMENTARY
- ANY_MAGNITUDE
- ANY_BIT
- ANY_STRING
- ANY_DATE
- ANY_NUM
- ANY_REAL
- ANY_INT
- LREAL

- REAL
- LINT
- DINT
- INT
- SINT
- ULINT
- UDINT
- UINT
- USINT
- TIME
- LWORD
- DWORD
- WORD
- BYTE
- BOOL
- STRING
- WSTRING
- DATE_AND_TIME
- DATE
- TIME_OF_DAY

Пример на оператор `hastype (variable: variable, type-spec):`

Предварительное условие: есть 2 приложения `App1` и `App2`. Переменная `g_multitype` объявлена в `App1` с типом LREAL и в приложении `App2` с типом STRING:

```
{IF (hastype (variable: g_multitype, LREAL))}
(* the following code line will be processed only in App1 *)
g_multitype := (0.9 + g_multitype) * 1.1;
{ELSIF (hastype (variable: g_multitype, STRING))}
(* the following code line will be processed only in App2 *)
g_multitype := 'this is a multitalent';
{END_IF}
```

hasvalue (define-ident, char-string)

Если определение (define-ident), определено, и у него есть указанное значение (char-string), то ее значение TRUE (ИСТИНА). Иначе она FALSE (ЛОЖЬ).

Пример на hasvalue (define-ident, char-string):

Precondition: Variable testis used in applications App1and App2. It gets value 1 in App1and value 2 in App2:

Предварительное условие: Переменная test используется в приложениях App1 и App2. Она получает значение 1 в App1 и значение 2 в App2:

```
{IF hasvalue(test,'1')}
(* the following code line will be processed in App1, because there var iable test
has value 1 *)
x := x + 1;
{ELSIF hasvalue(test,'2')}
(* the following code line will be processed in App1, because there var iable test
has value 2 *)
x := x + 2;
{END_IF}
```

NOT operator

Выражение получает значение TRUE (ИСТИНА), когда инвертированное значение оператора TRUE. operator может быть одним из операторов, описанных в этой главе.

Пример на NOT operator:

Предварительное условие: есть 2 приложения App1 и App2. POU PLC_PRG1 используется в App1 и App2. POU CheckBounds доступна только в App1:

```
{IF defined (pou: PLC_PRG1) AND NOT (defined (pou: CheckBounds))}
(* the following code line is only executed in App2 *)
bANDNotTest := TRUE;
{END_IF}
```

AND operator

Выражение получает значение TRUE (ИСТИНА), если оба оператора TRUE. operator может быть одним из операторов, перечисленных в этой таблице.

Пример на AND operator:

Предварительное условие: есть 2 приложения App1 и App2. POU PLC_PRG1 используется в применениях App1 и App2. POU CheckBounds доступна только в App1:

```
{IF defined (pou: PLC_PRG1) AND (defined (pou: CheckBounds))}
(* the following code line will be processed only in applications App1, because only
there "PLC_PRG1" and "CheckBounds" are defined *)
bANDTest := TRUE;
{END_IF}
```

OR operator

Выражение TRUE (ИСТИНА), если один из операторов TRUE. `operator` может быть одним из операторов, описанных в этой главе.

Example on `OR operator`: **Пример на** `OR operator`:

Предварительное условие: POU `PLC_PRG1` используется в приложениях `App1` и `App2`. POU `CheckBounds` доступна только в `App1`:

```
{IF defined (pou: PLC_PRG1) OR (defined (pou: CheckBounds))}
(* the following code line will be processed in applications App1 and A pp2, because both contain at least one of the POUs "PLC_PRG1" and "CheckBounds" *)
bORTest := TRUE;
{END_IF}
```

(operator)

Оператор скобок (`operator`) заключает в них `оператор`.

Раздел 27.5

Прагмы атрибутов

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Прагмы атрибутов	543
Задаваемые пользователем атрибуты	544
Атрибут `call_after_init`	546
Атрибут `displaymode`	547
Атрибут `ExpandFully`	548
Атрибут `global_init_slot`	549
Атрибут `hide`	550
Атрибут `hide_all_locals`	551
Атрибут `initialize_on_call`	552
Атрибут `init_namespace`	552
Атрибут `init_On_Onlchange`	553
Атрибут `instance-path`	554
Атрибут `linkalways`	555
Атрибут `monitoring`	556
Атрибут `namespace`	560
Атрибут `no_check`	562
Атрибут `no_copy`	562
Атрибут `no-exit`	563
Атрибут `no_init`	564
Атрибут `no_virtual_actions`	565
Атрибут `obsolete`	568
Атрибут `pack_mode`	569
Атрибут `qualified_only`	570
Атрибут `reflection`	570
Атрибут `subsequent`	571
Атрибут `symbol`	572
Атрибут `warning disable`	574

Прагмы атрибутов

Обзор

Вы можете назначить прагмы атрибутов *(см. стр. 530)* к надписи, чтобы влиять на компиляцию или предварительную компиляцию, которая является генерацией объектного кода.

Это определяемые пользователем атрибуты *(см. стр. 544)*, которые Вы можете использовать в сочетании с условными прагмами *(см. стр. 534)*.

Есть также следующие предопределенные стандартные прагмы атрибутов:

- attribute displaymode – атрибут displaymode *(см. стр. 547)*
- attribute ExpandFully – атрибут ExpandFully *(см. стр. 548)*
- attribute global_init_slot – атрибут global_init_slot *(см. стр. 549)*
- attribute hide – атрибут hide *(см. стр. 550)*
- attribute hide_all_locals – атрибут hide_all_locals *(см. стр. 551)*
- attribute initialize_on_call – атрибут initialize_on_call *(см. стр. 552)*
- attribute init_namespace – атрибут init_namespace *(см. стр. 552)*
- attribute init_On_Onlchange – атрибут init_On_Onlchange *(см. стр. 553)*
- attribute instance-path – атрибут instance-path *(см. стр. 554)*
- attribute linkalways – атрибут linkalways *(см. стр. 555)*
- attribute monitoring – атрибут monitoring *(см. стр. 556)*
- attribute namespace – атрибут namespace *(см. стр. 560)*
- attribute no_check – атрибут no_check *(см. стр. 562)*
- attribute no_copy – атрибут no_copy *(см. стр. 562)*
- attribute no-exit – атрибут no-exit *(см. стр. 563)*
- attribute noinit – атрибут noinit *(см. стр. 564*
- attribute no_virtual_actions – атрибут no_virtual_actions *(см. стр. 565*
- attribute obsolete – атрибут obsolete *(см. стр. 568)*
- attribute pack_mode – атрибут pack_mode *(см. стр. 569)*
- attribute qualified_only – атрибут qualified_only *(см. стр. 570)*
- attribute reflection – атрибут reflection *(см. стр. 570)*
- attribute subsequent – атрибут subsequent *(см. стр. 571)*
- attribute symbol – атрибут symbol *(см. стр. 572)*
- attribute warning disable – атрибут warning disable *(см. стр. 574)*

Задаваемые пользователем атрибуты

Обзор

Вы можете назначить произвольный задаваемые пользователями или определенные применением прагмы атрибутов на POU, декларацию типа или переменные. Этот атрибут может быть подвергнут запросу перед компиляцией условных прагм *(см. стр. 534)*.

Синтаксис

{attribute 'атрибут'}

Эта инструкция прагмы действительна для последующей декларации POU или декларации переменной.

Вы можете назначить задаваемый пользователем атрибут на:

- POU или действие
- переменную
- тип данных

Пример на POU и акции

Атрибут `vision` для функции `fun1`:

```
{attribute 'vision'}
FUNCTION fun1 :
INT VAR_INPUT
i : INT;
END_VAR
VAR
END_VAR
```

Пример на переменные

Атрибут `DoCount` для переменной `ivar` :

```
PROGRAM PLC_PRG
VAR
{attribute 'DoCount'};
ivar:INT;
bvar:BOOL;
END_VAR
```

Пример на типы

Атрибут `aType` для типа данных `DUT_1`:

```
{attribute 'aType'}
TYPE DUT_1 :
STRUCT
a:INT;
b:BOOL;
END_STRUCT
END_TYPE
```

Для использования условных прагм обращайтесь к главе *Условные прагмы* (*см. стр. 534*).

Атрибут `call_after_init`

Обзор

Используйте прагму `{attribute call_after_init}` для определения метода, который вызывают неявно после инициализации экземпляра функционального блока. С этой целью приложите атрибут и к самому функциональному блоку и к экземпляру метода, который вызывают (по причинам выполнения). Метод нужно вызвать после FB_Init *(см. стр. 525)*, и после применения значений переменных выражения инициализации в декларации экземпляра.

Эта функция поддержана начиная с SoMachine V3.0.

Синтаксис

{attribute 'attribute call_after_init'}

Пример

Со следующим определением:

```
{attribute 'call_after_init'}
FUNCTION_BLOCK FB
... <functionblock definition>
{attribute 'call_after_init'}
METHOD FB_AfterInit
... <method definition>
```

... декларация как следующее:

```
inst : FB := (in1 := 99);
```

... приведёт к следующему порядку выполнения кода:

```
inst.FB_Init();
inst.in1 := 99;
inst.FB_AfterInit();
```

Таким образом, в `FB_Afterinit` Вы можете воздействовать на задаваемую пользователем инициализацию.

Атрибут `displaymode`

Обзор

Используйте прагму `{attribute displaymode}` для определения режима работы монитора единственной переменной. Эта уставка перепишет глобальную уставку для режима работы монитора всех контролируемых переменных, сделанных через команды подменю **Display Mode** (по умолчанию в меню **Online**).

Поместите прагму в линию выше линии, содержащей декларацию переменной.

Синтаксис

{attribute 'displaymode':=<displaymode>}

Возможны следующие определения:

– для отображения в двоичном формате

```
{attribute 'displaymode':='bin'}
{attribute 'displaymode':='binary'}
```

– для отображения в десятичном формате

```
{attribute 'displaymode':='dec'}
{attribute 'displaymode':='decimal'}
```

– для отображения в шестнадцатеричном формате

```
{attribute 'displaymode':='hex'}
{attribute 'displaymode':='hexadecimal'}
```

Пример

```
VAR
  {attribute 'displaymode':='hex'}
  dwVar1: DWORD;
END_VAR
```

Атрибут `ExpandFully`

Обзор

Используйте прагму `{attribute 'ExpandFully'}`, чтобы сделать всех членов множеств используемыми в качестве входных переменных для визуализаций, на которые ссылаются, доступными в диалоговом окне **Visualization Properties**.

Синтаксис

{attribute 'ExpandFully'}

Example Пример

Визуализация `visu` предназначена, чтобы быть вставленной в структуру в рамках визуализации `visu_main`.

Поскольку входная переменная `arr` определена в интерфейсном редакторе `visu` и позже будет доступна для назначений в диалоговом окне **Properties** структуры в visu_main.

Чтобы получить доступные переменные компоненты множества в диалоговом окне **Properties**, вставьте атрибут `ExpandFully` в интерфейсный редактор `visu` непосредственно перед `arr`.

Декларация в интерфейсном редакторе `visu`:

```
VAR_INPUT
{attribute 'ExpandFully'}
arr : ARRAY[0..5] OF INT;
END_VAR
```

Получающееся диалоговое окно **Properties** структуры в `visu_main`:

Атрибут global_init_slot

Обзор

Вы можете применить прагму {attribute 'global_init_slot'} только для надписей. По умолчанию порядок инициализации переменных списка глобальных переменной произволен. Однако в некоторых случаях предписание порядка необходимо, когда переменные 1 глобального списка относятся к переменным другого списка. В этом случае Вы можете использовать прагму, чтобы установить порядок для глобальной инициализации.

Синтаксис

{attribute 'global_init_slot' := '< value>'}

Замените шаблон <value> целочисленным значением, описывающим порядок инициализации надписи. Значение по умолчанию 50,000. Меньшее значение вызывает более раннюю инициализацию. В случае надписей, несущих одинаковое значение для атрибута global_init_slot, последовательность их инициализации остается неопределенной.

Пример

Предположим, что проект включает 2 списка GVL_1 и GVL_2 глобальных переменных.

Глобальная переменная A является членом образцового списка глобальных переменных GVL_1:

```
{attribute 'global_init_slot' := '300'}
VAR_GLOBAL
A : INT:=1000;
END_VAR
```

Начальные значения переменных B и C GVL_2 зависят от переменной A.

```
{attribute'global_init_slot' := '350'}
VAR_GLOBAL
B : INT:=A+1;
C : INT:=A-1;
END_VAR
```

Установка атрибута 'global_init_slot' GVL_1 на 300, который является самым низким значением в образцовом порядке инициализации, помогает гарантировать, что выражение A+1 хорошо определено, инициализируя B.

Атрибут hide

Обзор

Прагма `{attribute hide}` помогает Вам препятствовать тому, чтобы переменные или даже целые надписи были показаны в функциональности компонентов листинга *(см. стр. 575)*, или помощника ввода или декларационной части в режиме онлайн. Будет скрыта только переменная, следующая за прагмой.

Синтаксис

{attribute 'hide'}

Чтобы скрыть все локальные переменные надписи, используйте атрибут hide_all_locals *(см. стр. 551)*.

Пример

Функциональный блок `myPOU` осуществлен, используя атрибут:

```
FUNCTION_BLOCK myPOU
VAR_INPUT
a:INT;
{attribute 'hide'}
a_invisible: BOOL;
a_visible: BOOL;
END_VAR
VAR_OUTPUT
b:INT;
END_VAR
```

В главной программе определены 2 экземпляра функционального блока `myPOU`:

```
PROGRAM PLC_PRG
VAR
POU1, POU2: myPOU;
END_VAR
```

При назначении входного значение POU1, функциональность листинга компонентов *(см. стр. 575)*, которая работает над печатью `POU1` в части реализации `PLC_PRG`, покажет входные переменные `a` и `a_visible` (и выходная переменная `b`). Скрытая входная переменная `a_invisible` не будет показана.

Атрибут hide_all_locals

Обзор

Прагма `{attribute 'hide_all_locals'}`, помогает Вам препятствовать тому, чтобы все локальные переменные надписи были показаны в функциональности компонентов листинга *(см. стр. 575)*, или помощника ввода. Этот атрибут идентичен назначению атрибута `hide` *(см. стр. 550)* к каждой детали локальных переменных.

Синтаксис

{attribute 'hide_all_locals'}

Пример

Функциональный блок myPOU осуществлен с использованием атрибута:

```
{attribute 'hide_all_locals'}
FUNCTION_BLOCK myPOU
VAR_INPUT
a:INT;
END_VAR
VAR_OUTPUT
b:BOOL;
END_VAR
VAR
c,d:INT;
END_VAR
```

В главной программе определены 2 функциональных блока `myPOU`:

```
PROGRAM PLC_PRG
VAR
POU1, POU2: myPOU;
END_VAR
```

Назначая входное значение для `POU1`, функциональность компонентов листинга *(см. стр. 575)*, которая работает над печатью `POU1` в реализационной части `PLC_PRG`, покажет переменные `a` и `b`. Скрытые местные переменные `c` или `d` не будут показаны.

Атрибут `initialize_on_call`

Обзор

Вы можете добавить прагму `{attribute initialize_on_call}`, чтобы ввести переменные. Вход функционального блока с этим атрибутом будет инициализирован при любом вызове функционального блока. Если вход ожидает указателя и если этот указатель был удален из-за онлайн изменения, то вход будет установлен в NULL.

Синтаксис

{attribute 'initialize_on_call'}

Атрибут `init_namespace`

Обзор

Переменная типа STRING или WSTRING, которая объявлена с прагмой `{attribute init_namespace}` в библиотеке будет инициализирована с текущим пространством имен этой библиотеки. Для получения дополнительной информации обратитесь к описанию управления библиотеками *(см. SoMachine, Functions and Libraries User Guide)*.

Синтаксис

{attribute 'init_namespace'}

Пример

Функциональный блок POU снабжён всеми необходимыми атрибутами:

```
FUNCTION_BLOCK POU
VAR_OUTPUT
{attribute 'init_namespace'}
myStr: STRING;
END_VAR
```

В рамках главной программы `PLC_PRG` определен экземпляр `fb` функционального блока `POU`:

```
PROGRAM PLC_PRG
VAR
fb:POU;
newString: STRING;
END_VAR
newString:=fb.myStr;
```

Переменная `myStr` будет инициализирована с текущим пространством имен, например `MyLib. XY`. Это значение будет назначено на `newString` в рамках главной программы.

Атрибут `init_On_Onlchange`

Обзор

Примените прагму `{attribute 'init_on_onlchange'}` для переменной, чтобы инициализировать эту переменную с каждым изменением онлайн.

Синтаксис

{attribute 'init_on_onlchange' }

Атрибут instance-path

Обзор

Вы можете добавить прагму `{attribute instance-path}` к локальной строковой переменной. Эта локальная строковая переменная будет инициализирована путём к **Applications tree** POU, которой принадлежит эта строковая переменная. Применение этой прагмы предполагает использование отражения атрибута *(см. стр. 570)* для соответствующего POU и дополнительного атрибута noinit *(см. стр. 564)* для строковой переменной.

Синтаксис

{attribute 'instance-path'}

Пример

Предположим, что следующий функциональный блок `POU` оборудован атрибутом `'reflection'`:

```
{attribute 'reflection'}
FUNCTION_BLOCK POU
VAR
{attribute 'instance-path'}
{attribute 'noinit'}
str: STRING;
END_VAR
```

В главной программе `PLC_PRG` вызывается экземпляр `myPOU` функционального блока `POU`:

```
PROGRAM PLC_PRG
VAR
myPOU:POU;
myString: STRING;
END_VAR
myPOU();
myString:=myPOU.str;
```

После инициализации экземпляра `myPOU`, строковой переменной `str` назначают путь экземпляра `myPOU`, например: `PLCWinNT.Application. PLC_PRG.myPOU`. Этот путь будет назначен на переменную `myString` в рамках главной программы.

ПРИМЕЧАНИЕ: длина строковой переменной может быть определена произвольно (даже > 255). Однако строка будет отрезана (от ее конца), если она будет назначена на более короткую переменную последовательность.

Атрибут `linkalways`

Обзор

Используйте прагму `{attribute 'linkalways'}`, чтобы отметить POU для компилятора таким способом, чтобы они были всегда включены в собираемую информацию. В результате объекты с этим выбором будут всегда собираться и загружаться в контроллер. Этот выбор затрагивает только POU и списки глобальных переменных (GVL), которые расположены ниже приложения или в библиотеках, которые вставлены ниже приложения. Опция компилятора **Link always** воздействует также.

Синтаксис

{attribute 'linkalways'}

Когда Вы используете редактор конфигурации символов, отмеченные POUs используются в качестве основания для выбираемых переменных для конфигурации символов.

Пример

Список глобальных переменных `GVLMoreSymbols` реализован с использованием атрибута `'linkalways'`:

```
{attribute 'linkalways'}
VAR_GLOBAS
g_iVar1: INT; g_iVar2: INT;
END_VAR
```

С этим кодом символы `GVLMoreSymbols` становятся выбираемыми в конфигурации символов **Symbol configuration**.

Редактор **Symbol configuration**

Атрибут monitoring

Обзор

Эта прагма атрибута позволяет Вам получать свойства и результаты вызова функции, наблюдаемые в онлайн виде редактора IEC или в списке наблюдения.

Мониторинг свойств

Добавьте прагму в линии выше определения свойства. Тогда имя, тип, и значение переменных свойства будут показаны в онлайн виде POU, использовающей свойство или в списке наблюдения. Там Вы можете также ввести подготовленные значения, чтобы форсировать переменные, принадлежащие свойству.

Пример свойства, подготовленного к мониторингу переменных

Пример вида мониторинга

Мониторинг текущего значения свойств переменных

Есть 2 различных способа наблюдать текущее значение свойств переменных. Для конкретного случая использования рассмотрите внимательно, какой признак подходит, чтобы на самом деле получить требуемое значение. Это будет зависеть от того, осуществлены ли операции на переменных внутри свойства:

1. Прагма {attribute'monitoring':=}'variable'

Неявная переменная создана для свойства, которая получит текущее значение каждый раз, когда приложение вызывает метод set (установки) или get (получения). Последнее значение, сохраненное в этой неявной переменной, будет наблюдаться.

Синтаксис

{attribute 'monitoring':='variable'}

2. Прагма {attribute 'monitoring':='call'}

Вы можете использовать этот атрибут только для свойств, возвращая простые типы данных или указатели, а не для структурированные типы.

Значение, которое будет наблюдаться, восстановлено прямым вызовом свойства: сервис мониторинга системы выполнения вызывает get метод и функция свойства будет выполнена.

ПРИМЕЧАНИЕ: Выбирая этот тип мониторинга вместо того чтобы использовать промежуточную переменную (*см. 1. Прагма*), рассмотрите возможные побочные эффекты, которые могут произойти, если какие-либо операции над переменной осуществляются в пределах свойства.

Синтаксис

{attribute 'monitoring':='call'}

Мониторинг результатов вызова функции

Вы можете использовать мониторинг вызова функции для любого постоянного значения, которое может интерпретироваться как 4-байтовое численное значение (например, INT, SHORT, LONG). Для других входных параметров (например, BOOL), используют переменную вместо постоянного параметра. Добавьте прагму `{attribute 'monitoring':='call'}` в линии выше декларирования функции. Тогда Вы можете контролировать эту переменную в редакторе текста в онлайн виде POU, в котором переменная получает результат вызова функции. Вы можете также добавить переменную к списку наблюдения для той же самой цели. Чтобы получить переменную немедленно в виде наблюдения, выполните команду **Add watchlist**.

Пример 1: Функции `FUN2` и `FUN_BOOL2` с атрибутом `'monitoring'`

Пример 2: Вызов функций `FUN2` и `FUN_BOOL2` в программе POU

```
PLC_PRG
1  PROGRAM PLC_PRG
2  VAR
3      nResult2, nResult3 : INT;
4      xResult2 : BOOL;
5      xResult4: BOOL;
6  END_VAR

1  // monitoring possible:
2  nResult2 := FUN2(64,GVL.VALUE2);
3  xResult2 := FUN_BOOL2(GVL.BOOLFALSE, GVL.BOOLTRUE);
4
5  // monitoring not possible (TRUE and FALSE cannot be interpreted directly)
6  FUN_BOOL2(TRUE, FALSE);
```

Пример 3: Вызовы функций в режиме онлайн:

PLC_PRG | FUN2 | FUN_BOOL2

Device.Application.PLC_PRG

Expression	Type	Value	Prepared value	Comment
nResult2	INT	32		
nResult3	INT	<???>		
xResult2	BOOL	FALSE		
xResult4	BOOL	<???>		

```
1  // Monitoring possible:
2  nResult2 [32] := FUN2 [???] (64,GVL.VALUE2 [???]);
3  nResult3 [???] := FUN3 [???] (64,GVL.VALUE2 [???] ,GVL.VALUE3 [???]);
4  xResult2 [FALSE] := FUN_BOOL2 [???] (GVL.BOOLFALSE [???] , GVL.BOOLTRUE [???]);
5  xResult4 [???] := FUN_BOOL4 [???] (GVL.BOOLTRUE [???] , GVL.BOOLFALSE [???], GVL.BOOL
6
7  // Monitoring not possible, True und FALSE cannot be Interpreted directly
8  FUN_BOOL2 [???] (TRUE, FALSE); [RETURN]
```

Watch 1

Expression	Type	Value	Prepared value	Comment
Device.Application.PLC_PRG.nResult2	INT	32		
Device.Application.PLC_PRG.xResult2	BOOL	FALSE		

Мониторинг переменных с неявным вызовом внешней функции

Для мониторинга переменных с неявным вызовом внешней функции должны быть выполнены следующие условия:

- Функция помечена `{attribute 'monitoring' := 'call'}`.
- Функция помечается как **Link Always**.
- Переменная помечена `{attribute 'monitoring_instead' := 'MyExternalFunction(a,b,c)'}`.
- Значения `a, b, c` являются целочисленными значениями и соответствуют входным параметрам функции, для вызова.

ПРИМЕЧАНИЕ: Форсирование или запись функций не поддерживаются. Вы можете неявно осуществить форсирование, добавив дополнительный входной параметр для конкретной функции, которая служит внутренним флагом форсирования.

ПРИМЕЧАНИЕ: Мониторинг функции не возможен для компактной системы выполнения.

Атрибут namespace

Обзор

В сочетании с символом атрибута *(см. стр. 572)*, прагма `{attribute namespace}` позволяет Вам переопределять пространство имен переменных проекта. Вы можете применить её на полные POU, как GVLs или программы, но не на конкретные переменные. Причастные переменные будут экспортироваться с новым определением пространства имен в файл символов и после загрузки этого файла будут доступными в контроллере.

Это также позволяет Вам доступ к переменным из POU или визуализаций, которые первоначально имели различные пространства имен. Например, это позволяет Вам выполнять предыдущую визуализацию SoMachine также в более поздней окружающей среде SoMachine.

Для получения дополнительной информации обратитесь к описанию конфигурации символов. Новый файл символов будет создан при загрузке или онлайн изменении проекта. Он загружается в контроллер вместе с приложением.

Синтаксис

{attribute 'namespace' := '<namespace>'}

Пример замены пространства имен для переменных программы

```
{attribute 'namespace':='prog'}
PROGRAM PLC_PRG
VAR
  {attribute 'symbol' := 'readwrite'}
  iVar:INT;
  bVar:BOOL;
END_VAR
```

Если к `iVar`, например, раньше был доступ через `App1. PLC_PRG.ivar`, теперь она доступна через `prog.ivar`.

Дальнейшие примеры замены

Оригинальное пространство имен	Переменная	Замена пространства имен	Доступ к переменной в рамках текущего проекта
App1.Lib2.GVL2	Var07	{attribute 'namespace':=''}	.Var07
App1.GVL2	Var02	{attribute 'namespace':='Ext''}	Ext.Var02
App1.GVL2.FB1	Var02	{attribute 'namespace':='App1.GVL2''}	App1.GVL2.Var02

Замены, показанные в таблице, приводят к следующим записям в файле символов:

```
<NodeList>
 <Node name="">
    <Node name="Var07" type="T_INT" access="ReadWrite">
 Node>
</NodeList>
<NodeList>
 <Node name="Ext">
    <Node name="Var02 " type="T_INT" access="ReadWrite"></Node>
 </Node>
</NodeList>
<NodeList>
 <Node name="App1">
  <Node name="GVL2">
   <Node name="Var02 " type="T_INT" access="ReadWrite"></Node>
    </Node>
 </Node>
</NodeList>
```

Атрибут `no_check`

Обзор

Прагма `{attribute no_check}` добавлена к POU, чтобы подавить требование любой POU для неявных проверок. Поскольку проверяющие функции могут влиять на работу, распространите этот атрибут на POU, которые часто вызываются или уже одобрены.

Синтаксис

{attribute 'no_check'}

Атрибут `no_copy`

Обзор

Обычно изменение онлайн потребует перераспределения экземпляров, например POU. Значение переменных в этом случае будет скопировано.

Если, однако, прагма `{attribute no_copy}` добавлена к переменной, онлайн копия изменения этой переменной не будет выполнена; вместо этого эта переменная будет инициализирована. Это может быть разумно в случае локальной переменной указателя, указывающей на переменную, на самом деле перемещенной изменением онлайн (и таким образом имеющей измененный адрес).

Синтаксис

{attribute 'no_copy'}

Атрибут `no-exit`

Обзор

Если функциональный блок предоставляет метод выхода *(см. стр. 528)*, Вы можете подавить его вызов для специального экземпляра с помощью назначения прагмы `{attribute no-exit}` экземпляру функционального блока.

Синтаксис

{attribute 'symbol'= 'no-exit'}

Пример

Предположим, что `exit` метод `FB_Exit`, был добавлен к функциональному блоку под именем POU:

В главной программе `PLC_PRG` 2 переменные типа POU иллюстрируются примерами:

```
PROGRAM PLC_PRG VAR
POU1 : POU;
{attribute 'symbol' := 'no-exit'}
POU2 : POU;
END_VAR
```

Когда переменная `bInCopyCode` становится TRUE в `POU1`, метод выхода `FB_Exit` вызывается, выходя из экземпляра, который будет скопирован впоследствии (изменение онлайн), хотя значение переменной `bInCopyCode` не будет иметь никакого влияния на `POU2`.

Атрибут `no_init`

Обзор

Переменные, которым предоставляют прагму `{attribute no_init}`, не будут инициализированы неявно. Прагма принадлежит переменной, объявленной впоследствии.

Синтаксис

{attribute 'no_init'}

также возможно

{attribute 'no-init'}

{attribute 'noinit'}

Пример

```
PROGRAM PLC_PRG
VAR
A : INT;
{attribute 'no_init'}
B : INT;
END_VAR
```

Если сброс выполняется в связанном приложении, переменная целого числа `A` будет снова инициализирована неявно 0, тогда как переменная B поддерживает значение, которое она в настоящее время имеет.

Атрибут `no_virtual_actions`

Обзор

Этот атрибут действителен для функциональных блоков, которые получены из основного функционального блока, осуществленного в SFC, и которые используют главный технологический процесс SFC базового класса. Действия, вызванные там, показывают то же самое виртуальное поведение как методы. Это означает, что действия базового класса могут быть отвергнуты определенными внедрениями, связанными с производными классами.

Чтобы помочь препятствовать тому, чтобы действие базового класса было отвергнуто, Вы можете назначить прагму `{attribute 'no_virtual_actions'}` для базового класса.

Синтаксис

{attribute 'no_virtual_actions'}

Пример

В следующем примере функциональный блок `POU_SFC` обеспечивает базовый класс, который будет расширен функциональным блоком `POU_child`.

При помощи ключевого слова `SUPER`, производный класс `POU_child` вызывает технологический процесс базового класса, который осуществлен в SFC.

Типичное внедрение этого технологического процесса ограничено начальным шагом. Он сопровождается 1 единственным шагом со связанным действием шага `ActiveAction`, касающимся назначения выходных переменных:

```
an_int:=an_int+1;           // counting the action calls
test_act:='father_action'; // writing string variable test_act
METH();          // Calling method METH for writing string variable test_meth
```

В случае производного класса `POU_child` действие шага будет переписано определенным внедрением `ActiveAction`. Оно отличается от оригинального за счет назначения строки `'child_action'` вместо `'father_action'` к переменной `test_act`.

Аналогично, метод `METH`, назначая последовательность `'father_method'` к переменной `test_meth` в базовом классе, будет переписан таким образом, что `test_meth` будет назначен на `'child_method'` вместо этого.

Главная программа `PLC_PRG` выполнит повторенные вызовы `Child` (экземпляр `POU_child`). Как ожидалось, фактическое значение выводимой строки сообщает о вызове действия и метода производного класса:

PLC_PRG [CoDeSys_SP_f_r_Win32: SPS-Logik: Applicat

CoDeSys_SP_f_r_Win32.Application.PLC_PRG

Expression	Type	Value
Child	POU_child	
test_meth	STRING	'child_method'
test_act	STRING	'child_action'
an_int	INT	53

Вы можете наблюдать отличное поведение, если базовому классу предшествует признак `'no_virtual_actions'`

```
{attribute 'no_virtual_actions'}
FUNCTION_BLOCK POU_SFC...
```

Принимая во внимание, что метод METH будет все еще переписан его реализацией в производном классе, вызов действия шага теперь приведет к вызову действия `ActiveAction` базового класса. Поэтому `test_act` получит значение `'father_action'`.

PLC_PRG [CoDeSys_SP_f_r_Win32: SPS-Logik: Applicat

CoDeSys_SP_f_r_Win32.Application.PLC_PRG

Expression	Type	Value
Child	POU_child	
test_meth	STRING	'child_method'
test_act	STRING	'father_action'
an_int	INT	204

Атрибут obsolete

Обзор

Вы можете добавить прагму устаревший (obsolete) к определению типа данных, чтобы вызвать определенную пользователями тревогу во время построения, если соответствующий тип данных (структура, функциональный блок, и так далее) используется в рамках проекта. Таким образом Вы можете объявить, что тип данных больше не используется.

В отличие от используемого в локальном масштабе прагмы сообщения *(см. стр. 532)*, эта тревога определена в рамках определения и таким образом глобальна для всех экземпляров типа данных.

Эта инструкция прагмы действительна для текущей линии или - если она помещена в отдельную линию - для последующей линии.

Синтаксис

{attribute 'obsolete' := 'user-defined text'}

Пример

Прагма устаревший вставлена в определение функционального блока fb1:

```
{attribute 'obsolete' := 'datatype fb1 not valid!'}
FUNCTION_BLOCK fb1
VAR_INPUT
i:INT;
END_VAR
...
```

Если fb1 используется в качестве типа данных в декларации, например, fbinst: fb1; следующее предупреждение будет выдано, когда проект будет построен:

```
'datatype fb1 not valid'
```

Атрибут pack_mode

Обзор

Прагма `{attribute 'pack_mode'}` определяет режим, которым структура данных упаковывается при размещении. Установите атрибут сверху структуры данных. Он влияет на упаковку целой структуры.

Синтаксис

{attribute 'pack_mode' := '<value>'}

Замените шаблон <value> залюченный в одиночные кавычки, одним из следующих доступных значений:

Value (Значение)	Назначенный режим пакета
0	выровненный (не будет никаких промежутков памяти),
1	1-байт-выровненный (идентичный выровненному)
2	2-байт-выровненный (максимальный размер промежутка памяти составляет 1 байт),
4	4-байт-выровненный (максимальный размер промежутка памяти составляет 3 байта),
8	8-байт-выровненный (максимальный размер промежутка памяти составляет 7 байт)

Пример

```
{attribute 'pack_mode' := '1'}
TYPE myStruct:
STRUCT
  Enable: BOOL;
  Counter: INT;
  MaxSize: BOOL;
  MaxSizeReached: BOOL;
END_STRUCT
END_TYPE
```

Переменная типа данных `myStruct` будет иллюстрироваться примерами выравнивания.

Если адрес его компонента `Enable` равен `0x0100`, то компонент `Counter` будет иметь адрес `0x0101`, `MaxSize` на `0x0103` и `MaxSizeReached` на `0x0104`.

При `pack_mode=2, Counter` был бы найден на `0x0102`, `MaxSize` на 0x0104 и `MaxSizeReached` на `0x0105`.

ПРИМЕЧАНИЕ: Вы можете также применить атрибут к POU. Используйте это применение тщательно из-за возможных существующих внутренних указателей POU.

Атрибут qualified_only

Обзор

Когда прагма `{attribute 'qualified_only'}` назначена сверху списка глобальных переменных, к переменным этого списка можно получить доступ только при помощи имени глобальной переменной, например `gvl.g_var`. Она работает даже на переменные типа перечисления. Может быть, полезно избежать несоответствия имени с локальными переменными.

Синтаксис

{attribute 'qualified_only'}

Пример

Предположим, что следующему списку глобальных переменных (GVL) предоставляют признак `'qualified_only'`:

```
{attribute 'qualified_only'}
VAR_GLOBAL
iVar:INT;
END_VAR
```

В POU `PLC_PRG` глобальную переменную нужно вызывать с префиксом `GVL`, как показано в этом примере:

```
GVL.iVar:=5;
```

Следующий незавершенный вызов переменной будет обнаружен как ошибка:

```
iVar:=5;
```

Атрибут reflection

Обзор

Прагма `{attribute 'reflection'}` присоединяется к надписям. Из-за исполнительных причин, это - обязательный признак для POU, несущей атрибут пути экземпляра *(см. стр. 554)*.

Синтаксис

{attribute 'reflection'}

Пример

Обратитесь к примеру `attribute instance-path` *(см. стр. 554)*.

Атрибут `subsequent`

Обзор

Прагма `{attribute 'subsequent'}` вынуждает переменные быть расположенными подряд в одном местоположении памяти. Если список изменится, целый список будет расположен в новом местоположении. Эта прагма используется в программах и списках глобальных переменных (GVL).

Синтаксис

{attribute 'subsequent'}

ПРИМЕЧАНИЕ: Если одна переменная в списке типа RETAIN, целый список будет расположен в сохраняемой памяти.

ПРИМЕЧАНИЕ: VAR_TEMP в программе с атрибутом `subsequent` будет обнаружено как ошибка компилятора.

Атрибут `symbol`

Обзор

Прагма `{attribute 'symbol'}` определяет, какие переменные должны быть обработаны в конфигурации символа.

Следующие экспортные операции выполнены на переменных:
- Переменные экспортируются как символы в список символов.
- Переменные экспортируются в файл XML в директории проекта.
- Переменные экспортируются в файл, не видимый и доступный на целевой системе для внешнего доступа, например, через OPC сервер.

Переменные, которым предоставляют этот атрибут, будут загружены на контроллер, даже если они не были сконфигурированы или не видимы в редакторе конфигурации символов.

ПРИМЕЧАНИЕ: Конфигурация символа должна быть доступной как объект ниже соответствующего применения в **Tools Tree**.

Синтаксис

{attribute 'symbol' := 'none' | 'read' | 'write' | 'readwrite'}

Доступ предоставлен только на символы, прибывающие из программ или списков глобальных переменных. Для доступа к символу определите имя символа полностью.

Вы можете назначить определение прагмы конкретным переменным или коллективно всем переменным, объявленным в программе.
- Чтобы быть действительными для единственной переменной, поместите прагму в линию перед декларацией переменной.
- Чтобы быть действительными для всех переменных, содержавшихся в части декларации программы, поместите прагму в первую линию редактора декларации. В этом случае Вы можете также изменить параметры настройки для конкретных переменных, явно добавив прагму.

Возможный доступ к символу определен следующими параметрами прагмы:
- `'none'` – нет доступа
- `'read'` - чтение
- `'write'` - запись
- `'readwrite'` – чтение/запись

Если никакой параметр не будет определен, по умолчанию будет действителен 'readwrite'.

Пример

Со следующей конфигурацией переменные `A` и `B` будут экспортированы с доступом по чтению и записи. Переменная `D` будет экспортироваться с доступом по чтению.

```
{attribute 'symbol' := 'readwrite'}
PROGRAM PLC_PRG
VAR
A : INT;
B : INT;
{attribute 'symbol' := 'none'}
C : INT;
{attribute 'symbol' := 'read'}
D : INT;
END_VAR
```

Атрибут warning disable

Обзор

Вы можете использовать прагму `warning disable`, чтобы подавить тревоги. Чтобы разрешить показ тревоги, используют прагму `warning restore`.

Синтаксис

{warning disable <compiler ID>}

У каждой тревоги и каждой ошибки, обнаруженной компилятором, есть уникальный ID, который показан в начале описания.

Пример сообщений компилятора

```
------ Build started: Application: Device.Application ------
typify code ...
C0196: Implicit conversion from unsigned Type 'UINT' to signed Type 'INT' : possible
change of sign
Compile complete -- 0 errors
```

Пример

```
VAR
      {warning disable C0195}
      test1 : UINT := -1;
      {warning restore C0195}
      test2 : UINT := -1;
END_VAR
```

В этом примере тревога будет обнаружена для `test2`. Но тревога не будет обнаружена для `test1`.

Раздел 27.6

Умная кодирующая функциональность

Умное кодирование

Обзор

Везде, где могут быть введены идентификаторы (как переменные или экземпляры функционального блока) (это может быть в редакторах языков IEC 61131-3 или в окнах **Watch, Trace, Visualization**), доступна умная кодирующая функциональность. Вы можете настроить эту особенность (активировано или деактивировано) в разделе **SmartCoding** диалогового окна **Tools** → **Options**.

Поддержка во вставке идентификатора

Умная кодирующая функциональность помогает вставить правильный идентификатор:

- Если Вы - в каком-либо месте, где может быть вставлен глобальный идентификатор - вставите точку (.) вместо идентификатора, покажется окно выбора. Оно перечисляет доступные в настоящее время глобальные переменные. Вы можете выбрать один из этих элементов и нажать клавишу RETURN, чтобы вставить его позади точки. Вы можете также вставить элемент, дважды щелкнув по записи в списке.
- Если Вы введёте экземпляр функционального блока или переменную структуры, сопровождаемую точкой (.), то появится окно выбора. Оно перечисляет переменные входа и выхода соответствующего функционального блока или компонентов структуры. Вы можете выбрать желаемый элемент, нажав клавишу RETURN или дважды щелкнув по записи в списке, чтобы вставить её.
- В ST редакторе, если Вы ввели какую-либо строку и нажимаете CTRL+SPACE, покажется окно выбора. Оно перечисляет списки POU и глобальные переменные, доступные в проекте. Первая запись списка, которая начинается с заданной строки, будет выбрана. Нажмите клавишу RETURN, чтобы вставить его в программу.

Примеры

Умная кодирующая функциональность предлагает компоненты структуры:

Умная кодирующая функциональность предлагает компоненты функционального блока:

Глава 28

Типы данных

Содержание этой главы

Эта глава содержит следующие разделы

Раздел	Тема	Страница
28.1	Общая информация	578
28.2	Стандартные типы данных	579
28.3	Расширения к стандарту IEC	582
28.4	Определяемые пользователем типы данных	590

Раздел 28.1

Общая информация

Типы данных

Обзор

Вы можете при программировании в SoMachine использовать стандартные типы данных (*см. стр. 579*), определяемые пользователями типы данных (*см. стр. 590*), или экземпляры функциональных блоков. Каждому идентификатору назначается тип данных. Этот тип данных диктует, сколько места в памяти будет зарезервировано и какие значения она хранит.

Раздел 28.2

Стандартные типы данных

Стандартные типы данных

Обзор

SoMachine поддерживает все типы данных *(см. стр. 578)*, описанные стандартом IEC61131-3.

В этой главе описаны следующие типы данных:

- BOOL логический *(см. стр. 579)*
- Integer целый *(см. стр. 579)*
- REAL / LREAL вещественный *(см. стр. 580)*
- STRING строковый *(см. стр. 581)*
- Time Data Types типы времени и даты *(см. стр. 581)*

Кроме того, поддержаны некоторые расширяющие стандарт типы данных *(см. стр. 582)*, и Вы можете определить свои собственные определяемые пользователем типы данных *(см. стр. 590)*.

BOOL

У переменных типа BOOL могут быть значения, TRUE (1) и FALSE (0). Зарезервированы 8 битов места в памяти.

Для получения дополнительной информации обращайтесь к разделу *Константы BOOL* (*см. стр. 711*).

ПРИМЕЧАНИЕ: Вы можете использовать неявные проверки, чтобы утвердить преобразование типов переменных (обратитесь к главе POU для неявных проверок *(см. стр. 173)*),

Целое число

Таблица перечисляет доступные типы данных целого числа. Каждый из типов покрывает различный диапазон значений. Применяются следующие ограничения диапазона.

Тип данных	Нижний предел	Верхний предел	Объём памяти
BYTE	0	255	8 bit
WORD	0	65,535	16 bit
DWORD	0	4,294,967,295	32 bit
LWORD	0	2^{64}-1	64 bit
SINT	–128	127	8 bit
USINT	0	255	8 bit
INT	–32,768	32,767	16 bit
UINT	0	65,535	16 bit

Тип данных	Нижний предел	Верхний предел	Объём памяти
DINT	–2,147,483,648	2,147,483,647	32 bit
UDINT	0	4,294,967,295	32 bit
LINT	-2^{63}	$2^{63}-1$	64 bit
ULINT	0	$2^{64}-1$	64 bit

ПРИМЕЧАНИЕ: Преобразования от больших типов к меньшим типам могут привести к потере информации. Для получения дополнительной информации обратитесь к описанию числовых констант *(см. стр. 717)*.

ПРИМЕЧАНИЕ: Вы можете использовать неявные проверки, чтобы утвердить преобразование переменных типов (обратитесь к главе POU для неявных проверок *(см. стр. 173)*).

REAL / LREAL

Типы данных REAL и LREAL являются так называемыми типами с плавающей запятой. Они представляют рациональные числа. 32 бита места в памяти зарезервированы для REAL и 64 бита для LREAL.

Диапазон значений для REAL:

1.401e-45...3.403e+38

Диапазон значений для LREAL:

2.2250738585072014e-308...1.7976931348623158e+308

ПРИМЕЧАНИЕ: Поддержка типа данных LREAL зависит от целевого устройства. Посмотрите в соответствующей документации, преобразовывается ли 64-битный тип LREAL в REAL во время компиляции (возможно с потерей информации) или сохраняется.

ПРИМЕЧАНИЕ: Если REAL или LREAL преобразуются в SINT, USINT, INT, UINT, DINT, UDINT, LINT или ULINT и значение действительного числа вне диапазона значений этого целого числа, результат будет неопределенным и будет зависеть от целевой системы. Даже в этом случае возможно исключение. Чтобы получить код, независимый от целевой платформы, поддерживайте какие-либо превышения с помощью приложения. Если число REAL/LREAL будет в диапазоне целочисленного значения, преобразование будет работать для всех систем одинаковым образом.

При присвоении `i1: = r1;` обнаружена ошибка. Поэтому предыдущее примечание применяется при использовании операторов преобразования *(см. стр. 663)*, как следующее:

```
i1 := REAL_TO_INT(r1);
```

Для получения дополнительной информации обращайтесь к константам (операндам) REAL/LREAL *(см. стр. 718)*.

ПРИМЕЧАНИЕ: Вы можете использовать неявные проверки, чтобы утвердить преобразование переменных типов (обратитесь к главе *POU для неявных проверок (см. стр. 173)*).

STRING

Переменная типа данных STRING может содержать любую строку символов. Размер записи в декларации определяет место в памяти, которое будет зарезервировано для переменной. Это относится к количеству знаков в строке и может быть помещено в круглые скобки или квадратные скобки. Если не будет задана никакая спецификация размера, будет использоваться размер по умолчанию 80 символов.

Обычно длина строки не ограничена. Но строковые функции могут обработать только последовательности с длиной 1... 255 символов. Если переменная будет инициализироваться слишком длинной строкой для типа данных переменной, последовательность будет соответственно урезана справа налево.

ПРИМЕЧАНИЕ: место в памяти, необходимое для переменной типа STRING, составляет 1 байт на символ + 1 дополнительный байт. Это означает, декларация "STRING[80]" требует 81 байт.

Пример декларации строки с 35 символами:

```
str:STRING(35):='This is a String';
```

Для получения дополнительной информации обратитесь к константам (операндам) WSTRING *(см. стр. 584)*, и STRING *(см. стр. 719)*.
ПРИМЕЧАНИЕ: Вы можете использовать неявные проверки, чтобы утвердить преобразование переменных типов (обратитесь к главе *POUs для неявных проверок (см. стр. 173)*).

Типы данных времени

Типы данных TIME, TIME_OF_DAY (сокращенно TOD), DATE и DATE_AND_TIME (сокращенно DT) поддерживаются внутренне как DWORD. Время дано в миллисекундах в TIME и TOD. Время в TOD начинается в 12:00. Время дано в секундах в DATE и DT, начиная с 1 января 1970 в 12:00.

Для получения дополнительной информации обратитесь к следующим описаниям:

- Data Types типы данных *(см. стр. 578)*
- LTIME: расширение к стандарту IEC 61131-3, доступное как тип данных времени с 64 битами *(см. стр.583)*
- TIME constants константы времени *(см. стр. 712)*
- DATE constants константы даты *(см. стр. 714)*
- DATE_AND_TIME constants константы даты и времени *(см. стр. 715)*
- TIME_OF_DAY constants константы времени суток *(см. стр. 716)*

ПРИМЕЧАНИЕ: Вы можете использовать неявные проверки, чтобы утвердить преобразование типов переменных (обратитесь к главе *POUs для неявных проверок (см. стр. 173)*),

Раздел 28.3

Расширения к стандарту IEC

Обзор

Этот раздел перечисляет типы данных, которые поддержаны SoMachine в дополнение к стандарту IEC 61131-3.

Содержатие этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
UNION	583
LTIME	583
WSTRING	584
BIT	584
References - Ссылки	585
Pointers - Указатели	587

UNION

Обзор

Как расширение к стандарту IEC 61131-3, Вы можете объявить объединения (unions) в задаваемых пользователем типах.

Компоненты объединения имеют одно и то же смещение. Это означает, что они занимают то же самое место хранения. Таким образом, принимая определение союза, как показано в следующем примере, назначение `name.a` также управляет `name.b`.

Пример

```
TYPE name: UNION
a : LREAL;
b : LINT;
END_UNION
END_TYPE
```

LTIME

Обзор

Как расширение IEC 61131-3, LTIME поддержан как основа времени для таймеров с высоким разрешением. LTIME имеет размер 64 бита и разрешение наносекунд.

Синтаксис

LTIME#<time declaration>

The time declaration can include the time units as used with the TIME constant and as:

Декларация времени может включать единицы времени, как используется с константой TIMEЮ и как:

- us : microseconds мкс: микросекунды
- ns : nanoseconds нс: наносекунды

Пример

```
LTIME1 := LTIME#1000d15h23m12s34ms2us44ns
```

Сравните с размером TIME 32 бита и разрешением миллисекунд *(см. стр. 581)*.

WSTRING

Обзор

Этот строковый тип данных является расширением к стандарту IEC 61131-3.

Он отличается от стандартного типа STRING (ASCII) интерпретацией в формате Unicode и необходимостью 2 байт для каждого символа и 2-байт дополнительного места в памяти (только 1 в случае STRING).

Пример

```
wstr:WSTRING:="This is a WString";
```

Для получения дополнительной информации обратитесь к следующим описаниям:

- STRING строка *(см. стр. 581)*
- STRING constants константы STRING *(см. стр.719)* (операкнды)

BIT

Обзор

Вы можете использовать тип данных BIT только для конкретных переменных в структурах (Structures) *(см. стр. 595)*. Возможные значения TRUE (1) и FALSE (0).

Элемент BIT используетет 1 бит места в памяти и позволяет Вам обращаться к адресоваться к отдельным битам структуры по имени (для получения дополнительной информации, обратитесь к параграфу *Доступ к битам в структурах (см. стр. 596)*). Битовые элементы, которые объявлены один за другим, будут объединены в байтах. В отличие от типов BOOL *(см. стр. 579)*, где 8 битов зарезервированы в любом случае, использование места в памяти может быть оптимизировано. С другой стороны, доступ к битам занимает определенно больше времени. По этой причине используйте тип данных BIT, если Вы хотите сохранить несколько булевых фрагментов информации в компактном формате.

References Ссылки

Обзор

Этот тип данных доступен в расширении стандарта IEC 61131-3.

Ссылка (reference) является псевдонимом для объекта. Псевдоним может быть написан или прочитан через идентификаторы. Отличие от указателя (pointer) - то, что указываемое значение непосредственно затрагивается и что назначение ссылки и значения фиксировано. Установите адрес ссылки через отдельную операцию назначения. Вы можете использовать оператор `_ISVALIDREF`, чтобы проверить, что ссылка указывает на верное значение (которое не равно 0). Для получения дополнительной информации обратитесь к параграфу *Проверка на верные ссылки* далее ниже в этой главе.

Синтаксис

<identifier> : REFERENCE TO <data type>

Пример декларации

```
ref_int : REFERENCE TO INT;
a : INT;
b : INT;
```

`ref_int` теперь доступен для использования в качестве псевдонима для переменных типа INT.

Пример использования

```
ref_int REF= a;        (* ref_int now points to a *)
ref_int := 12;         (* a now has value 12 *)
b := ref_int * 2;      (* b now has value 24 *)
ref_int REF= b;        (* ref_int now points to b *)
ref_int := a / 2;      (* b now has value 6 *)
ref_int REF= 0;        (* explicit initialization of the reference *)
```

ПРИМЕЧАНИЕ: невозможно объявить ссылки как REFERENCE TO REFERENCE (ссылка на ссылку) или ARRAY OF REFERENCE (множество ссылок) или POINTER TO REFERENCE (указателя на ссылку).

Проверка для верность ссылок

Вы можете использовать оператор `_ ISVALIDREF`, чтобы проверить указывает ли ссылка на верное значение, которое является значением, не равным 0.

Синтаксис

```
<boolean variable> := _ISVALIDREF(identifier, declared with type <REFERENCE TO
<datatype>);
```

```
<boolean variable> будет TRUE, если ссылка указывает на верное значение, FALSE если
нет.
```

Пример

Объявление

ivar : INT;

ref_int : REFERENCE TO INT;

ref_int0: REFERENCE TO INT;

testref: BOOL := FALSE;

Реализация

ivar := ivar +1; ref_int REF= hugo;

ref_int0 REF= 0;

testref := ISVALIDREF(ref_int); (* will be TRUE, because ref_int points to ivar, which is unequal 0 *)

testref0 := ISVALIDREF(ref_int0); (* will be FALSE, because ref_int0 is set to 0 *)

Указатели

Обзор

Как расширение стандарта IEC 61131-3, Вы можете использовать указатели.

Указатели хранят адреса переменных, программ, функциональных блоков, методов и функций во время выполнения программы приложения. Указатель может указать на любой из этих объектов и на любой тип данных *(см. стр. 578)*, даже на определяемые пользователем типы данных *(см. стр. 591)*. Возможность использования функции неявной проверки указателя описана ниже в параграфе *Функция CheckPointer (см. стр. 589)*.

Синтаксис декларации указателя

<identifier>: POINTER TO <data type | function block | program | method | function>;

Разыменование указателя означает получение значения, в настоящее время хранившемуся по адресу, на который он указывает. Вы можете разыменовать указатель, добавляя оператор ^ содержания (ASCII знак вставки или символ циркумфлекса) *(см. стр. 660)* после идентификатора указателя. См. pt^ в примере ниже.

Вы можете использовать оператор адреса `ADR` *(см. стр. 659)* для назначения адреса переменной к указателю.

Пример

```
VAR
 pt:POINTER TO INT;         (* of pointer pt *)
var_int1:INT := 5;          (* declaration of variables var_int1 and var_int2 *)
var_int2:INT;
END_VAR
pt := ADR(var_int1); (* address of var_int1 is assigned to pointer pt *)
var_int2:= pt^;  (* value 5 of var_int1 gets assigned to var_int2 via dereferencing
of pointer pt; *)
```

Указатели функции

SoMachine также поддерживает указатели функций. Эти указатели могут быть переданы во внешние библиотеки, но невозможно вызвать указатель функции в приложении в системе программирования. Функция выполнения регистрации возврата из вызова функций (системная функция библиотеки) ожидает указатель функции, и, в зависимости от отзыва, для которого требовали регистрацию, соответствующая функция будет вызвана неявно системой во время выполнения (например, при STOP). Чтобы позволить такой системный вызов (система поддержки выполнения), установите соответствующие свойства (по умолчанию под **View** → **Properties...**→**Build**) для объекта функции.

Вы можете использовать оператор ADR *(см. стр. 659)* в именах функций, именах программ, именах функциональных блоков и именах методов. Так как функции могут переместиться после онлайн изменения, результат не адрес функции, а адрес указателя на функцию. Этот адрес действителен, пока функция существует на цели.

Выполнение команды **Online Change** может изменить содержание адресов.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕДОПУСТИМЫЙ УКАЗАТЕЛЬ

При использовании указателей на адреса и выполняя команду **Online Change**, всегда проверяйте содержание указателей.

Отказ следовать этим инструкциям может привести к повреждению оборудования или травме.

Доступ через индекс к указателям

Как расширение к стандарта IEC 61131-3, предоставлен индексный [ ] доступ к переменным типа POINTER, STRING *(см. стр. 581)* и WSTRING *(см. стр. 584)*.

Если индексный доступ используется для переменной типа указателя, смещение pint[i] равняется (pint + i * SIZEOF(base type))^. Индексный доступ также выполняет неявное разыменование указателя. Тип результата - основной тип указателя.

Обратите внимание, что `pint[7] не равняется (pint + 7)^`.

- Если индексный доступ используется для переменной типа STRING, результатом является символ по смещению `index-expr`. Результат – BYTE. `str[i]` будет возвращать i-ый символ строки, как SINT (ASCII).
- Если индексный доступ используется на переменной типа WSTRING, результатом является символ по смещению. Результат имеет тип WORD. `wstr[i]` будет возвращать i-ый символ строки, как INT (Unicode).

ПРИМЕЧАНИЕ: Вы можете также использовать ссылки (References) *(см. стр. 585)*. В отличие от указателя, ссылки непосредственно затрагивают значение.

Функция CheckPointer

Для проверки доступа указателя во время выполнения Вы можете использовать доступную неявно функцию проверки `CheckPointer`. Ее вызывают перед каждым доступом к адресу указателя. Чтобы достигнуть этого, добавьте в приложение объект **POUs for implicit checks**. Сделайте это, активировав флажок, связанный с типом **CheckPointer**, выбирая язык реализации и подтвердив Ваши параметры настройки нажатием на **Open**. Это открывает проверочную функцию в редакторе, соответствующем выбранному языку реализации. Независимо от этого выбора задана декларационная часть. Вы не можете изменить ее кроме добавления дальнейших локальных переменных. Однако в отличие от других проверочных функций, нет никакой доступной по умолчанию реализации `CheckPointer`.

ПРИМЕЧАНИЕ: нет никакого неявного вызова проверочной функции для указателя `THIS`.

Шаблон:

Декларационная часть:

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckPointer : POINTER TO BYTE
VAR_INPUT
ptToTest : POINTER TO BYTE;
iSize : DINT;
iGran : DINT;
bWrite: BOOL;
END_VAR
```

Реализующая часть: (неполная):

```
// No standard way of implementation. Fill your own code here
CheckPointer := ptToTest;
```

При вызове, функции предоставляются следующие входные параметры:

- `ptToTest`: Целевой адрес указателя
- `iSize`: Размер переменной, на которую ссылаются; тип данных `iSize` должен быть совместим с целым числом и должен покрыть максимальный потенциальный размер данных, сохраненный по адресу указателя.
- `iGran`: Степень детализации доступа, которая является самым большим типом неструктурированных данных, используемым в переменной, на которую ссылаются; тип данных `iGran` должен быть совместим с целым числом
- `bWrite`: Тип доступа (TRUE = доступ по записи, FALSE = доступ по чтению); тип данных `bWrite` должен быть BOOL.

В случае положительного результата проверки неизмененный входной указатель будет возвращен (`ptToTest`).

Раздел 28.4

Задаваемые пользователем типы данных

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Определяемые типы данных	591
Arrays – Массивы	592
Structures – Структуры	595
Enumerations – Перечисления	597
Типы поддиапазона	599

Определяемые типы данных

Обзор

В дополнение к стандартным типам данных Вы можете определить специальные типы данных в рамках проекта.

Вы можете определить их через создание DUT (Data Unit Type - Тип единицы данных) объектов в **POUs tree** или **Devices tree** или в декларационной части POU.

Смотрите рекомендации по наименованию объектов *(см. стр. 500)*, чтобы сделать его максимально уникальным.

Смотрите следующие, определяемые пользователем, типы данных:

- arrays — массивы *(см. стр. 592)*
- structures — структуры *(см. стр. 595)*
- enumerations — перечисления *(см. стр. 597)*
- subrange types — типы поддиапазона *(см. стр. 599)*
- references — ссылки *(см. стр. 585)*
- pointers — указатели *(см. стр. 587)*

Arrays – Массивы

Обзор

Одно-, двух- и трехмерные области (массивы) поддержаны как элементарные типы данных. Вы можете определить множества и в части декларации POU и в списках глобальных переменных. Вы можете также использовать неявные проверки границ *(см. стр. 593)*.

Синтаксис

<Array_Name>:ARRAY [<ll1>..<ul1>,<ll2>..<ul2>,<ll3>..<ul3>] OF <elem. Type>

`ll1, ll2, ll3` определяют нижний предел диапазона поля.

`ul1, ul2` и `ul3` определяют верхний предел диапазона поля.

Значения диапазона должны иметь тип целого числа.

Пример

```
Card_game: ARRAY [1..13, 1..4] OF INT;
```

Инициализация массивов

Пример для полной инициализации массива

```
arr1 : ARRAY [1..5] OF INT := [1,2,3,4,5];
arr2 : ARRAY [1..2,3..4] OF INT := [1,3(7)]; (* short for 1,7,7,7 *)
arr3 : ARRAY [1..2,2..3,3..4] OF INT := [2(0),4(4),2,3];
       (* short for 0,0,4,4,4,4,2,3 *)
```

Пример инициализации множества структуры

Определение структуры

```
TYPE STRUCT1
STRUCT
 p1:int;
 p2:int;
 p3:dword;
END_STRUCT
END_TYPE
```

Инициализация массива

```
ARRAY[1..3] OF STRUCT1:= [(p1:=1,p2:=10,p3:=4723),(p1:=2,p2:=0,p3:=299)
,(p1:=14,p2:=5,p3:=112)];
```

Пример частичной инициализации массива

```
arr1 : ARRAY [1..10] OF INT := [1,2];
```

Элементы, которым не предписано никакое значение, инициализируются начальным значением по умолчанию основного типа. В предыдущем примере элементы arr1 [6]... arr1[10] поэтому инициализированы 0.

Доступ к компонентам массива

В двумерном массиве получите доступ к компонентам следующим образом:

<Array-Name>[Index1,Index2]

Пример:

```
Card_game [9,2]
```

Функции проверки границ массива

Чтобы правильно получить доступ к элементу массива во время выполнения, функция `CheckBounds` должна быть доступна приложению. Поэтому добавьте объект **POUs for implicit checks** к приложению, используя **Add Object** → **POUs for implicit checks**. Выберите флажок, связанный с типом **CheckBounds**. Выберите язык реализации. Подтвердите свои параметры настройки, нажав **Open**. Функция `CheckBound` будет открыта в редакторе, соответствующем выбранному языку реализации. Независимо от этого выбора задана декларационная часть. Вы не можете изменить её за исключением добавления дальнейших локальных переменных. ST редактор дает предложение реализации функции по умолчанию, которое Вы можете изменить.

Эта функция проверки должна рассматривать граничные нарушения соответствующим методом (например, устанавливая обнаруженный ошибоный флаг или регулируя индекс). Функция будет вызвана неявно, как только назначена переменная типа ARRAY.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННОЕ ДЕЙСТВИЕ ОБОРУДОВАНИЯ

Не изменяйте декларационную часть неявной функции проверки.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

Пример использования функции CheckBounds

Реализация по умолчанию функции проверки следующая:

Декларационная часть

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckBounds : DINT
VAR_INPUT
index, lower, upper: DINT;
END_VAR
```

Реализующая часть

```
// Implicitly generated code : Only an Implementation suggestion
IF index < lower THEN
CheckBounds := lower;
ELSIF index > upper THEN
CheckBounds := upper;
ELSE
CheckBounds := index;
END_IF
```

При вызове функция получает следующие входные параметры:

- `index`: индекс элемента области
- `lower`: нижний предел диапазона области
- `upper`: верхний предел диапазона области

Пока индекс находится в диапазоне, возвращаемое значение - сам индекс. Иначе в соответствии с нарушением диапазона или верхний, или нижний предел диапазона поля будут возвращены.

Превышение верхнего предела массива a

Верхний предел массива `a` превышен в следующем примере:

```
PROGRAM PLC_PRG
VAR a: ARRAY[0..7] OF BOOL;
b: INT:=10;
END_VAR
a[b]:=TRUE;
```

В этом случае, неявный вызов функции `CheckBounds` предшествует действию присвоения так, что значения индекса изменено с 10 на верхний предел 7. Поэтому значение TRUE будет присвоено элементу `a[7]` из массива. Это - то, как Вы можете исправить предпринятый доступ вне диапазона области через функцию `CheckBounds`.

Structures – Структуры

Обзор

Создавайте структуры в проекте как DUT (Data Type Unit) объекты через диалоговое окно **Add Object**.

Они начинаются с ключевых слов `TYPE` и `STRUCT` и заканчиваются `END_STRUCT` и `END_TYPE`.

Синтаксис

```
TYPE <structurename>:
STRUCT
  <declaration of variables 1>
  ...
  <declaration of variables n>
END_STRUCT
END_TYPE
```

<structurename> - тип, который распознаётся всюду в проекте и может использоваться как и стандартный тип данных.

Вложенные структуры разрешены. Единственное ограничение - то, что переменные не могут быть присвоены адресам (объявление `AT` не разрешено).

Пример

Пример для определения структуры под именем `Polygonline`:

```
TYPE Polygonline:
STRUCT
   Start:ARRAY [1..2] OF INT;
   Point1:ARRAY [1..2] OF INT;
   Point2:ARRAY [1..2] OF INT;
   Point3:ARRAY [1..2] OF INT;
   Point4:ARRAY [1..2] OF INT;
End:ARRAY [1..2] OF INT;
END_STRUCT END_TYPE
```

Инициализация структур

Пример:

```
Poly_1:polygonline := ( Start:=[3,3], Point1:=[5,2], Point2:=[7,3], Point3:=[8,5],
Point4:=[5,7], End:= [3,5]);
```

Инициализации с переменными не возможны. Для примера инициализации массива структуры обратитесь к *Массивам* (*Arrays*) *(см. стр. 592)*.

Доступ к компонентам структуры

Вы можете получить доступ к компонентам структуры, используя следующий синтаксис:

<structurename>.<componentname>

Для предыдущего примера структуры `Polygonline` Вы можете получить доступ к компоненту `Start` через `Poly_1.Start`.

Битовый доступ в структурах

Тип данных BIT *(см. стр. 584)* является специальным типом данных, который может быть определен только в структурах. Он использует пространство памяти в 1 бит и позволяет Вам адресовать одиночные биты структуры по имени.

```
TYPE <structurename>:
STRUCT
  <bitname bit1> : BIT;
  <bitname bit2> : BIT;
  <bitname bit3> : BIT;
  ...
  <bitname bitn> : BIT;
END_STRUCT
END_TYPE
```

Вы можете получить доступ к BIT компоненту структуры при помощи следующего синтаксиса:

<structurename>.<bitname>

ПРИМЕЧАНИЕ: использование ссылок и указателя на BIT переменные не возможно. Кроме того, BIT переменные не позволены в массивах.

Enumerations – Перечисления

Обзор

Перечисление - определяемый пользователем тип, который составлен из многих строковых констант. Эти константы упоминаются как перечисляемые величины.

Перечисляемые величины распознаются глобально во всех областях проекта, даже если они объявлены в POU.

Перечисление создается в проекте как объект DUT через диалоговое окно **Add Object**.

ПРИМЕЧАНИЕ: Локальное объявление перечисления возможно только в TYPE.

Синтаксис

TYPE <identifier> (<enum_0>,<enum_1>, ...,<enum_n>) |<base data type>;END_TYPE

Переменная типа <identifier> может взять на себя одно из перечисляемых значений <enum_..> и будет инициализировано первым. Эти значения совместимы с целыми числами, что означает, что Вы можете выполнить операции с ними, как Вы сделали бы с целочисленными переменными. Вы можете присвоить номер x переменной. Если величины перечисления не будут инициализированы определенными значениями в объявлении, подсчет начнется с 0. При инициализации гарантируйте, что начальные значения увеличиваются в строке компонентов. Верность числа проверяется в то время, когда это выполняется.

Пример

```
TYPE TRAFFIC_SIGNAL: (red, yellow, green:=10);     (* The initial value for each of
the colors is red 0, yellow 1, green 10 *)
END_TYPE
TRAFFIC_SIGNAL1 : TRAFFIC_SIGNAL;
TRAFFIC_SIGNAL1:=0; (* The value of the traffic signal is "red" *)
FOR i:= red TO green DO i := i + 1;
END_FOR;
```

Первое расширение стандарта IEC 61131-3

Вы можете использовать тип имени перечислений (как оператор объема *(см. стр. 707)*) для снятия неоднозначности доступа к константы перечисления.

Поэтому возможно использовать одну и ту же константу в различных перечислениях.

Пример

Определение двух перечислений

```
TYPE COLORS_1: (red, blue);
END_TYPE
TYPE COLORS_2: (green, blue, yellow);
END_TYPE
```

Использование величины перечисления, blue в POU

Объявление

```
colorvar1 : COLORS_1;
colorvar2 : COLORS_2;
```

Реализация

```
(* possible: *)
colorvar1 := colors_1.blue;
colorvar2 := colors_2.blue;
(* not possible: *)
colorvar1 := blue;
colorvar2 := blue;
```

Второе расширение стандарта IEC 61131-3

Вы можете определить явно базовый тип данных перечисления, которое по умолчанию является INT.

Пример

Базовым типом данных для перечисления `BigEnum` должен быть DINT.

```
TYPE BigEnum : (yellow, blue, green:=16#8000) DINT;
END_TYPE
```

Типы поддиапазона

Обзор

Тип поддиапазона - определяемый пользователем тип *(см. стр. 591)*, чей диапазон значений - только подмножество того из типа исходных данных. Вы можете также использовать неявные проверки границ диапазона *(см. стр. 600)*.

Вы можете сделать объявление в DUT объекте, но Вы можете также объявить переменную непосредственно с типом поддиапазона.

Синтаксис

Синтаксис для объявления как DUT объекта:

TYPE <name>: <Inttype> (<ug>..<og>) END_TYPE;

`<name>`	допустимый IEC идентификатор
`<inttype>`	один из типов данных SINT, USINT, INT, UINT, DINT, UDINT, BYTE, WORD, DWORD (LINT, ULINT, LWORD)
`<ug>`	константа, совместимая с основным типом, установка самой нижней границы диапазона типов. Нижняя граница сама включена в этот диапазон.
`<og>`	константа, совместимая с основным типом, установка самой верхней границы диапазона типов. Верхняя граница сама включена в этот диапазон.

Пример

```
TYPE
  SubInt : INT (-4095..4095);
END_TYPE
```

Прямое объявление переменной с типом поддиапазона

```
VAR
  i : INT (-4095..4095);
  ui : UINT (0..10000);
END_VAR
```

Если значение присвоено типу поддиапазона (в объявлении или в реализации), но не соответствует этому диапазону (например, i:= 5000 в верхнем показанном примере объявления), будет выдано сообщение.

Функции проверки для границ диапазона

Чтобы проверить пределы диапазона во время выполнения, функции `CheckRangeSigned` или `CheckRangeUnsigned` должны быть доступны приложению. Вы можете добавить объект **POUs for implicit checks** для неявных проверок к приложению, используя диалоговое окно **Add Object**. Отметьте флажок, связанный с типом **CheckRangeSigned** или **CheckRangeUnsigned**. Выберите язык реализации. Подтвердите свои параметры настройки, нажав **Open**. Выбранная функция будет открыта в редакторе, соответствующем выбранному языку реализации. Независимо от этого выбора задан раздел описаний (деклараций). Вы не можете изменить его за исключением добавления дальнейших локальных переменных. Редактор ST предлагает реализацию по умолчанию функции, которую Вы можете изменить.

Цель этой функции проверки - надлежащая обработка нарушений поддиапазона (например, устанавливая флаг ошибки или изменяя значение). Функция вызывается неявно, как только будет присвоена переменная типа поддиапазона.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННОЕ ДЕЙСТВИЕ ОБОРУДОВАНИЯ

Не изменяйте декларационную часть неявной функции проверки.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

Пример

Присвоение переменной, принадлежащей типу поддиапазона со знаком, влечет за собой неявный вызов `CheckRangeSigned`. Реализация по умолчанию этой функции, обрезающей значение к допустимому диапазону, обеспечена следующим образом:

Декларационная часть:

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckRangeSigned : DINT
VAR_INPUT
value, lower, upper: DINT;
END_VAR
```

Реализующая часть:

```
// Implicitly generated code : Only an Implementation suggestion
IF (value < lower) THEN
CheckRangeSigned := lower;
ELSIF(value > upper) THEN
CheckRangeSigned := upper;
ELSE
CheckRangeSigned := value;
END_IF
```

При вызове функция получает следующие входные параметры:

- `value`: значение, которое будет присвоено типу диапазона
- `lower`: нижняя граница диапазона
- `upper`: верхняя граница диапазона

Пока присвоенное значение находится в допустимом диапазоне, оно будет использоваться в качестве возвращаемого значения функции. Иначе, в соответствии с нарушением диапазона, будут возвращены или верхняя, или нижняя граница диапазона.

Присвоение `i: = 10*y` будет теперь неявно заменено на

```
i := CheckRangeSigned(10*y, -4095, 4095);
```

Если `y`, например, есть значение 1000, переменной `i` будет присвоено не 10*1000=10000 (в соответствии с исходной реализацией), а верхняя граница диапазона, которой является 4095.

То же применяется к функции `CheckRangeUnsigned`.

ПРИМЕЧАНИЕ: Если ни одна из функций, `CheckRangeSigned` или `CheckRangeUnsigned` не присутствуют, никакая проверка типа типов поддиапазона не происходит во время выполнения. В этом случае переменная `i` может получить любое значение между –32768 и 32767 в любое время.

Глава 29

Инструкции по программированию

Содержание этой главы

Эта глава содержит следующие разделы:

Раздел	Тема	Страница
29.1	Соглашение о присвоении имен	604
29.2	Префиксы	606

Раздел 29.1

Соглашение о присвоении имен

Общая информация

Создание имен обозначений

Выберите соответствующее, короткое описание на английском языке для каждого обозначения (designator): базисное имя. Базисное имя должно быть самообъясняющим. Используйте заглавную (прописную) первую букву каждого слова в базисном имени. Записывайте остальное в строчных буквах (пример: FileSize). Это базисное имя получает префиксы, чтобы указать объем и свойства.

Каждый раз, когда возможно, обозначение не должно содержать больше 20 символов. Это значение - рекомендация. При необходимости Вы можете скорректировать число вверх или вниз.

Если используются сокращения стандартных терминов (TP, JK-FlipFlop, ...), имя не должно содержать больше 3 прописных букв в последовательности.

Чувствительность к регистру

Примите во внимание, что чувствительность к регистру (использование и верхнего и нижнего), особенно для префиксов, улучшает удобочитаемость при использовании обозначений в IEC программе.

ПРИМЕЧАНИЕ: компилятор не чувствителен к регистру.

Допустимые буквы

Используйте только следующие буквы, числа и специальные символы в указателях:

0...9, A...Z, a...z,

Чтобы быть в состоянии вывести на экран префиксы четко, символ подчеркивание используется в качестве разделителя. Синтаксис объяснен в соответствующем разделе о префиксе.

Не используйте символ подчеркивания в базисном имени.

Примеры

Рекомендуемое обозначение	Нерекомендуемые обозначения
diState	diSTATE
xInit	x_Init
diCycleCounter	diCyclecounter
lrRefVelocity	lrRef_Velocity
c_lrMaxPosition	clrMaxPosition
FC_PidController	FC_PIDController

Раздел 29.2

Префиксы

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Части префикса	607
Порядок префиксов	608
Префикс области	610
Префикс типа данных	611
Префикс свойства	613
Префикс POU	614
Префикс пространства имен	615

Части префикса

Обзор

Префиксы используются, чтобы присвоить имена по функции.

Доступны следующие части префиксов:

Часть префикса	Использование	Синтаксис	Пример
scope prefix префикс области *(см. стр. 610)*	область переменных и констант	[scope]_[designator]	`G_diFirstUserFault`
data type prefix префикс типа данных *(см. стр. 611)*,	идентификация типа данных переменных и констант	[type][designator]	`xEnable`
property prefix префикс свойства *(см. стр. 613)*,	идентификация свойств переменных и констант	[property]_[designator]	`c_iNumberOfAxes`
POU prefix префикс POU *(см. стр. 614)*,	идентифиґция, если POU былґ реализованґ как функция, функциональный блок или программа	[POU]_[designator]	`FB_VisuController`
namespace prefix префикс пространства имен *(см. стр. 615)*	для POU, типов данных, переменных и констант, объявленых в библиотеке	[namespace].[identifier]	`TPL.G_dwErrorCode`

Порядок префиксов

Обзор

Обозначения содержат префикс области и префикс типа. Используйте префикс свойства согласно свойству переменных (например, для констант). Дополнительный префикс пространства имен используется для библиотек.

Обязательный порядок

Следующий порядок является обязательным:

[scope][property][_][type][identifier]

Префиксы области и префиксы свойства отделены от префиксов типа символом подчеркивания (_).

Пример

```
Gc_dwErrorCode   : DWORD;
diCycleCounter   : DINT;
```

Дополнительный префикс пространства имен используется для библиотек:

```
[namespace].[scope][property][_][type][identifier]
```

Пример

```
ExampleLibrary.Gc_dwErrorCode
```

Независимые организационные модули программы (POU)

Вставьте символ подчеркивания, чтобы разделить префиксы организационных модулей программы (функции, функциональные блоки и программы) от идентификаторов:

[POU][_][identifier]

Пример

```
FB_MotionCorrection
```

Используйте дополнительный префикс пространства имен для библиотек:

[namespace].[POU][_][identifier]

Префиксы пространства имен отделены от префиксов POU точкой (.).

Пример

```
ExampleLibrary.FC_SetError()
```

Зависимые организационные модули программы (POU)

Методы (method), действия (action) и свойства (property) рассматриваются как зависимые POU. Они используются на уровне ниже независимого POU.

У методов и действий нет префиксов.

Свойства получают префикс типа своего возвращаемого значения.

Пример

```
PROPERTY lrVelocity : LREAL
```

Префикс области

Обзор

Префикс области указывает область переменных и констант. Он указывает, является ли это локальной или глобальной переменной или константой.

Глобальные переменные обозначены прописной G_ и префикс свойства `c` добавляют к глобальным константам (в каждом случае сопровождается символом подчеркивания).

ПРИМЕЧАНИЕ: Дополнительно идентифицируйте глобальные переменные и константы библиотек с пространством имен библиотеки.

Префикс области	Тип	Использование	Пример
no prefix	`VAR`	локальная переменная	`xEnable`
`G_`	`VAR_GLOBAL`	глобальная переменна	`G_diFirstUserFault`
`Gc_`	`VAR_GLOBAL CONSTANT`	глобальная константа	`Gc_dwErrorCode`

Пример

```
VAR_GLOBAL CONSTANT
  Gc_dwExample : DWORD := 16#0000001A;
END_VAR
```

Доступ к глобальной переменной библиотеки с пространством имен INF:

```
INF.G_dwExample := 16#0000001A;
```

Префикс типа данных

Стандартные типы данных

Префикс типа данных идентифицирует тип данных переменных и констант.

ПРИМЕЧАНИЕ: префикс типа данных может также быть составным объектом, например, для указателей, ссылок и массивов. Указатель или массив перечисляются сначала, сопровождаются префиксом типа массива или типа указателя.

Префиксы типа данных стандарта IEC 61131-3, а также префиксы для расширений стандарта перечислены в таблице.

Префикс типа данных	Тип	Использование (ячейки памяти)	Пример
x	BOOL	булевая переменная (8 бит)	xName
by	BYTE	последовательность битов (8 бит)	byName
w	WORD	последовательность битов (16 бит)	wName
dw	DWORD	последовательность битов (32 бит)	dwName
lw	LWORD	последовательность битов (64 бит)	lwName
si	SINT	короткое целое (8 бит)	siName
i	INT	целое число (16 битов)	iName
di	DINT	двойное целое число (32 бита)	diName
li	LINT	длинное целое (64 бита)	liName
uli	ULINT	длинное целое без знака (64 бита)	uliName
usi	USINT	короткое целое без знака (8 битов)	usiName
ui	UINT	целое число без знака (16 битов)	uiName
udi	UDINT	двойное целое число без знака (32 бита)	udiName
r	REAL	число с плавающей точкой (32 бита)	rName
lr	LREAL	длинное число с плавающей точкой (64 бита)	lrName
dat	DATE	дата (32 бита)	datName
t	TOD	время (32 бита)	tName
dt	DT	дата и время (32 бита)	dtName
tim	TIME	продолжительность (32 бита)	timName
ltim	LTIME	продолжительность (64 бита)	ltimName
s	STRING	символьная строка ASCII	sName
ws	WSTRING	символьная строка unicode	wsName
p	pointers	указатель	pxName
r	reference	ссылка	rxName
a	array	область	axName
e	enumeration	тип списка	eName

Префикс типа данных	Тип	Использование (ячейки памяти)	Пример
st	struct	структура	stName
if	interface	интерфейс	ifMotion
ut	union	объединение	uName
fb	function block	функциональный блок	fbName

Примеры

```
piCounter: POINTER TO INT;
aiCounters: ARRAY [1..22] OF INT;
paiRefCounter: POINTER TO ARRAY [1..22] OF INT;
apstTest       : ARRAY[1..2] OF POINTER TO ST_MotionStructure;
rdiCounter     : REFERENCE TO DINT;
ifMotion       : IF_Motion;
```

Префикс свойства

Обзор

Префикс свойства идентифицирует свойства переменных и констант.

Тип префикса	Использование	Синтаксис	Пример
c_	VAR CONSTANT	локальная константа	c_xName
r_	VAR RETAIN	сохраняемая переменная типа retain	r_xName
p_	VAR PERSISTENT	сохраняемая переменная типа persistent	p_xName
rp_	VAR PERSISTENT	сохраняемая переменная типа retain persistent	rp_xName
i_	VAR_INPUT	входной параметр POU	i_xName
q_	VAR_OUTPUT	выходной параметр POU	q_xName
iq_	VAR_IN_OUT	параметр входа/выхода POU	iq_xName
ati_	AT %IX x.y AT %IB z AT %IW k	входная переменная, которая должна быть записана на входной области IEC	ati_x0_0MasterEncoderInitOK
atq_	AT %QX x.y AT %QB z AT %QW k	выходная переменная, которая должна быть записана на выходной области IEC	atq_w18AxisNotDone
atm_	AT %MX x.y AT %MB z AT %MW k	переменная маркера, которая должна быть записана на области маркеров IEC	atm_w19ModuleNotReady

ПРИМЕЧАНИЕ:

- Не объявляйте константы, как RETAIN или PERSISTENT.
- Не объявляйте какие-либо RETAIN переменные внутри POU. Это назначит сохранить всю POU в области сохраняемой (retain) памяти.

Пример AT-объявленных переменных

Имя AT-объявленной переменной также содержит тип целевой переменной. Он используется как префикс типа.

```
ati_xEncoderInit AT %IX0.0 : BOOL;
atq_wAxisNotDone AT %QW18 : WORD;
atm_wModuleNotReady AT %MW19 : WORD;
```

ПРИМЕЧАНИЕ: переменная также может быть размещена на адрес в отображающемся диалоговом окне устройства в конфигурации контроллера (редактор устройства). Предлагает ли устройство это диалоговое окно, описано в его документации.

Префикс POU

Обзор

Следующие организационные модули программы (POU) определены в IEC 61131-3:

- function — функция
- function block — функциональный блок
- program — программа
- data structure — структура данных
- list type — тип списка
- union — объединение
- interface — интерфейс

Обозначение состоит из префикса POU и максимально короткого имени (например, FB_GetResult). Точно так же, как и для переменных, используйте прописную первую букву каждого слова в базисном имени. Остальное записывайте в строчных буквах. Формируйте составное имя POU из глагола и существительного.

Префикс записывается с символом подчеркивания перед именем и идентифицирует тип POU на основе таблицы:

Префикс POU	Тип	Применение	Пример
SR_	PROGRAM	программа	SR_FlowPackerMachine
FB_	FUNCTION_BLOCK	функциональные блоки	FB_VisuController
FC_	FUNCTION	функции	FC_SetUserFault
ST_	STRUCT	структура данных	ST_StandardModuleInterface
ET_	Enumeration	тип списка	ST_StandardModuleInterface
UT_	UNION	объединение	UT_Values
IF_	INTERFACE	интерфейс	IF_CamProfile

Префикс пространства имен

Обзор

Вы можете просмотреть пространство имен библиотеки в менеджере библиотек **(Library Manager)**. Используйте короткий акроним (PacDriveLib -> `PDL`) как пространство имен. Не изменяйте для библиотеки пространство имен по умолчанию.

Чтобы зарезервировать однозначное пространство имен для Ваших собственных, разработанных самостоятельно библиотек, свяжитесь со своим ответственным Schneider Electric.

Пример

Функция `FC_DoSomething ()` расположена в библиотеке TestlibraryA (пространство имен `TLA`), а также в TestlibraryB (пространство имен `TLB`). К соответствующей функции получают доступ, снабжая префиксом пространства имен.

Если обе библиотеки расположены в проекте, следующий вызов приводит к ошибке, обнаруженной во время компиляции:

```
FC_DoSomething();
```

In this case, it is necessary to define clearly which POU is to be called up.

В этом случае необходимо четко определить, какая POU должна быть вызвана.

```
TLA.FC_DoSomething();
TLB.FC_DoSomething();
```

Глава 30

Операторы

Обзор

SoMachine поддерживает все IEC операторы. В отличие от стандартных функций, эти операторы распознаются неявно всюду в проекте.

Помимо IEC операторов, поддерживаются следующие операторы, которые не предписаны стандартом:

- `ANDN`
- `ORN`
- `XORN`
- `SIZEOF` (обратитесь к арифметическим операторам *(см. стр. 618)*),
- `ADR`
- `BITADR`
- оператор содержания (обратитесь к адресным операторам *(см. стр. 658)*,
- некоторые операторы области *(см. стр. 706)*,

Содержание этой главы

Эта глава содержит следующие разделы:

Раздел	Тема	Страница
30.1	Арифметические операторы	618
30.2	Операторы битовых строк	632
30.3	Операторы сдвига битов	637
30.4	Операторы выбора	645
30.5	Операторы сравнения	651
30.6	Операторы адреса	658
30.7	Оператор вызова	662
30.8	Операторы преобразования типов	663
30.9	Числовые функции	680
30.10	IEC Extending Operators Операторы расширения IEC	693
30.11	Оператор инициализации	708

Раздел 30.1

Арифметические операторы

Обзор

Доступны следующие, предписанные стандартом IEC1131-3, операторы:

- ADD
- MUL
- SUB
- DIV
- MOD
- MOVE

Кроме того, есть следующий расширяющий стандарт оператор:

- SIZEOF

Рассмотрите возможные переполнения арифметических операций в случае, если получающееся значение превышает диапазон типа данных, используемого для переменной результата. Это может привести к низким значениям, записанным в машину вместо высоких значений, или наоборот.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННОЕ ДЕЙСТВИЕ ОБОРУДОВАНИЯ

Всегда проверяйте операнды и результаты, используемые в математических операциях, чтобы избежать арифметического переполнения.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

Содержание этого раздела

Этот раздел содержит слендующие темы:

Тема	Страница
ADD	619
MUL	621
SUB	623
DIV	625
MOD	628
MOVE	630
SIZEOF	631

ADD

Обзор

IEC оператор для сложения переменных.

Разрешённые типы

- BYTE
- WORD
- DWORD
- LWORD
- SINT
- USINT
- INT
- UINT
- DINT
- UDINT
- LINT
- ULINT
- REAL
- LREAL
- TIME
- TIME_OF_DAY(TOD)
- DATE
- DATE_AND_TIME(DT)

Для типов данных времени возможны следующие комбинации:

- TIME+TIME=TIME
- TOD+TIME=TOD
- DT+TIME=DT

В редакторе FBD/LD оператор ADD - расширяемый блок. Это означает, что вместо последовательно соединённых блоков ADD, Вы можете использовать один блок с многочисленными входами. Используйте команду **Insert Input** для добавления дополнительных входов. Число неограничено.

Пример в IL

```
LD      7
ADD     2
ADD     4
ADD     7
ST      iVar
```

Пример в ST

```
var1 := 7+2+4+7;
```

Пример в FBD

1. последовательность блоков ADD
2. расширенный блок ADD
3. блок ADD с параметрами EN/ENO

MUL

Обзор

IEC оператор для умножения переменных

Разрешённые типы

- BYTE
- WORD
- DWORD
- LWORD
- SINT
- USINT
- INT
- UINT
- DINT
- UDINT
- LINT
- ULINT
- REAL
- LREAL
- TIME

Переменные TIME могут быть перемножены с целочисленными переменными.

В редакторе FBD/LD оператор MUL - расширяемый блок. Это означает, что вместо последовательно связанных блоков MUL, Вы можете использовать один блок с многочисленными входами. Используйте команду **Insert Input** для добавления дополнительных входов. Число неограничено.

Пример в IL

```
LD      7
MUL     2      ,
        4      ,
        7
ST      Var1
```

Пример в ST

```
var1 := 7*2*4*7;
```

Пример в FBD

1. последовательность блоков `MUL`
2. расширенный блок `MUL`
3. блок `MUL` с параметрами EN/ENO

SUB

Обзор

IEC оператор для вычитания одной переменной из другой.

Разрешённые типы

- BYTE
- WORD
- DWORD
- LWORD
- SINT
- USINT
- INT
- UINT
- DINT
- UDINT
- LINT
- ULINT
- REAL
- LREAL
- TIME
- TIME_OF_DAY(TOD)
- DATE
- DATE_AND_TIME(DT)

Для типов данных времени возможны следующие комбинации:

- TIME–TIME=TIME
- DATE–DATE=TIME
- TOD–TIME=TOD
- TOD–TOD=TIME
- DT-TIME=DT
- DT-DT=TIME

Обратите внимание, что отрицательные значения TIME не определены.

Пример в IL

```
LD      7
SUB     2
ST      Var1
```

Пример в ST

```
var1 := 7-2;
```

Пример в FBD

DIV

Обзор

IEC оператор для деления одной переменной на другую:

Разрешённые типы

- BYTE
- WORD
- DWORD
- LWORD
- SINT
- USINT
- INT
- UINT
- DINT
- UDINT
- LINT
- ULINT
- REAL
- LREAL
- TIME

Переменные TIME могут быть разделены на целочисленные переменные.

Пример в IL

(Результат в Var1 равен 4.)

```
LD      8
DIV     2
ST      Var1
```

Пример в ST

```
var1 := 8/2;
```

Пример в FBD

1. последовательность блоков DIV
2. единственный блок DIV
3. блок DIV с параметрами EN/ENO

Различные целевые системы могут вести себя по-разному относительно ошибки деления на нуль. Это может привести к останову (HALT) контроллера или может пройти необнаруженным.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННОЕ ДЕЙСТВИЕ ОБОРУДОВАНИЯ

Используйте функции проверки, описанные в этом документе, или Ваши собственные проверки, чтобы избежать деления на нуль в коде программы.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

ПРИМЕЧАНИЕ: Для получения дополнительной информации о неявных функциях проверки, обращайтесь к главе *POUs для неявных проверок* *(см. стр. 173)*.

Функции проверки

Вы можете использовать следующие функции проверки, чтобы проверить значение делителя во избежание деления на 0 и адаптировать их при необходимости:

- `CheckDivInt`
- `CheckDivLint`
- `CheckDivReal`
- `CheckDivLReal`

Для получения информации о вставке функции обратитесь к описанию функции **POUs для неявных проверок** *(см. стр. 173)*.

Функции проверки вызваются автоматически, перед каждым найденным делением в коде приложения. Смотрите следующий пример для реализации функции `CheckDivReal`.

Реализация по умолчанию функции CheckDivReal

Раздел описания

```
// Implicitly generated code : DO NOT EDIT
FUNCTION CheckDivReal : REAL
VAR_INPUT
 divisor:REAL;
END_VAR
```

Раздел реализации:

```
// Implicitly generated code : only an suggestion for implementation
IF divisor = 0 THEN
 CheckDivReal:=1;
ELSE
 CheckDivReal:=divisor;
END_IF;
```

Оператор `DIV` использует вывод функции `CheckDivReal` как делитель. В следующем примере деление на 0 запрещено, поскольку инициализированное нулевое значение делителя `d` изменено на 1 в `CheckDivReal`, до выполнения деления. Поэтому результат деления 799.

```
PROGRAM PLC_PRG
VAR
 erg:REAL;
 v1:REAL:=799;
 d:REAL;
END_VAR
erg:= v1 / d;
```

MOD

Обзор

IEC оператор для деления одной переменной на другую с остатком.

Разрешённые типы:

- BYTE
- WORD
- DWORD
- LWORD
- SINT
- USINT
- INT
- UINT
- DINT
- UDINT
- LINT
- ULINT

Результат этой функции - целочисленный остаток от деления.

Различные целевые системы могут вести себя по-разному относительно ошибки деления на нуль. Это может привести к останову (HALT) контроллера или может пройти необнаруженной.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННОЕ ДЕЙСТВИЕ ОБОРУДОВАНИЯ

Используйте функции проверки, описанные в этом документе, или Ваши собственные проверки, чтобы избежать деления на нуль в коде программы.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

ПРИМЕЧАНИЕ: Для получения дополнительной информации о неявных функциях проверки, обращайтесь к главе *POUs для неявных проверок* (*см. стр. 173*).

Пример в IL

Результат в Var1 равен 1.

```
LD      9
MOD     2
ST      Var1
```

Пример в ST

```
var1 := 9 MOD 2;
```

Пример в FBD

MOVE

Обзор

IEC оператор для присвоения переменной значения другой переменной надлежащего типа данных. Оператор `MOVE` возможен для всех типов данных.

Поскольку `MOVE` доступен как блок в графических редакторах FBD, LD, CFC, там функциональность (разблокирование), EN/ENO также может быть применена для присвоения переменной.

Пример в CFC в сочетании с функцией EN/ENO

Только если `en_i` равна TRUE, `var1` будет присвоена `var2`.

Пример в IL

Результат: var2 получает значение var1

```
LD        var1
MOVE
ST        var2
```

Вы получаете тот же результат в последовательности операторов

```
LD        var1
ST        var2
```

Пример в ST

```
ivar2 := MOVE(ivar1);
```

Вы получаете тот же результат при

```
ivar2 := ivar1;
```

SIZEOF

Обзор

Этот арифметический оператор не определен стандартом IEC 61131-3.

Вы можете использовать его, чтобы определить число байтов, требуемых для данной переменной `x`.

Оператор `SIZEOF` возвращает значение без знака. Тип возвращаемого значения будет адаптирован к найденному размеру переменной `x`.

Возвращаемое значение `SIZEOF(x)`	Тип данных константы, неявно используемой для найденного размера
0 ≤ размер x < 256	USINT
256 ≤ размер x < 65,536	UINT
65,536 ≤ размер x < 4,294,967,296	UDINT
4,294,967,296 ≤ размер x	ULINT

Пример в ST

```
var1 := SIZEOF(arr1);        (* d.h.: var1:=USINT#10; *)
```

Пример в IL

Результат равен 10

```
arr1:ARRAY[0..4] OF INT; Var1:INT;
LD        arr1
SIZEOF
ST        Var1
```

Раздел 30.2

Операторы битовых строк

Обзор

Доступны следующие операторы строки битов, соответствующие стандарту IEC1131-3:

- AND *(см. стр. 633)*
- OR *(см. стр. 634)*
- XOR *(см. стр. 635)*
- NOT *(см. стр. 636)*

Следующие операторы не определены стандартом и не доступны:

- ANDN
- ORN
- XORN

Операторы строки битов сравнивают соответствующие биты двух или нескольких операндов.

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
AND	633
OR	634
XOR	635
NOT	636

AND

Обзор

IEC оператор строки битов для поразрядного AND битовых операндов.

Если из входных битов каждый равняется 1, то получающийся бит будет 1, иначе 0.

Разрешённые типы:

- BOOL
- BYTE
- WORD
- DWORD
- LWORD

Пример в IL

Результат в `Var1` равен 2#1000_0010.

```
Var1:BYTE;
LD        2#1001_0011
AND       2#1000_1010
ST        var1
```

Пример в ST

```
var1 := 2#1001_0011 AND 2#1000_1010
```

Пример в FBD

OR

Обзор

IEC оператор строки битов для поразрядного "ИЛИ" битовых операндов.

Если, по крайней мере, 1 из входных битов будет равняться 1, получающийся бит будет 1, иначе 0.

Разрешённые типы:

- BOOL
- BYTE
- WORD
- DWORD
- LWORD

Пример в IL

Результат в `var1` равен 2#1001_1011.

```
var1:BYTE;
LD        2#1001_0011
OR        2#1000_1010
ST        Var1
```

Пример в ST

```
Var1 := 2#1001_0011 OR 2#1000_1010
```

Пример в FBD

XOR

Обзор

IEC оператор строки битов для поразрядного XOR битовых операндов.

Если только 1 из входных битов будет равняться 1, то получающийся бит будет 1; если оба или ни один будут 1, получающийся бит будет 0.

Разрешённые типы:

- BOOL
- BYTE
- WORD
- DWORD
- LWORD

ПРИМЕЧАНИЕ: XOR позволяет добавлять дополнительные входы. Если доступны больше чем 2 входа, то операция "исключающее ИЛИ" выполняется на первых 2 входах. Результат, в свою очередь, будет объединен по XOR, с входом 3, и т.д. Это имеет эффект, что нечетное число входов приведет к результирующему биту = 1.

Пример в IL

Результат равен 2#0001_1001.

```
Var1:BYTE;
LD        2#1001_0011
XOR       2#1000_1010
ST        var1
```

Пример в ST

```
Var1 := 2#1001_0011 XOR 2#1000_1010
```

Пример в FBD

NOT

Обзор

IEC оператор строки битов для поразрядного "НЕ" битового операнда.

Результирующий бит будет 1, если соответствующий входной бит 0 и наоборот.

Разрешённые типы

- BOOL
- BYTE
- WORD
- DWORD
- LWORD

Пример в IL

Результат в `Var1` равен 2#0110_1100.

```
Var1:BYTE;
LD        2#1001_0011
NOT
ST        var1
```

Пример в ST

```
Var1 := NOT 2#1001_0011
```

Пример в FBD

Раздел 30.3

Операторы сдвига битов

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
SHL	638
SHR	640
ROL	641
ROR	643

SHL

Обзор

IEC оператор для поразрядного сдвига влево операнда.

`erg:= SHL (in, n)`

`in`: операнд, который будет смещен влево

`n`: число битов, на которые `in` смещается влево

ПРИМЕЧАНИЕ: Если `n` превышает ширину типа данных, зависит от целевой системы, как BYTE, WORD, DWORD и LWORD будут заполнены. Некоторые вызывают заполнение нулями (0), другие - `n MOD <register width>`.

ПРИМЕЧАНИЕ: количество битов, которое рассматривется для арифметической операции, зависит от типа данных входной переменной. Если входная переменная - константа, рассматривают самый маленький возможный тип данных. Тип данных выходной переменной не имеет никакого эффекта вообще на арифметическую операцию.

Примеры

Рассмотрите в следующем примере в шестнадцатеричной нотации различные результаты для `erg_byte` и `erg_word`. Результат зависит от типа данных входной переменной (BYTE или WORD), несмотря на то, что значения входных переменных `in_byte` и `in_word` являются одинаковыми.

Пример в ST

```
PROGRAM shl_st
VAR
 in_byte : BYTE:=16#45; (* 2#01000101 )
 in_word : WORD:=16#0045; (* 2#0000000001000101 )
 erg_byte : BYTE; erg_word : WORD;
 n: BYTE :=2;
END_VAR
erg_byte:=SHL(in_byte,n); (* Result is 16#14, 2#00010100 *)
erg_word:=SHL(in_word,n); (* Result is 16#0114, 2#0000000100010100 *)
```

Пример в FBD

Пример в IL

```
LD      in_byte
SHL     2
ST      erg_byte
```

SHR

Обзор

IEC оператор для поразрядного сдвига вправо операнда.

`erg:= SHR (in, n)`

`in`: операнд, который будет смещен вправо

`n`: число битов, на которые `in` смещается вправо

ПРИМЕЧАНИЕ: Если `n` превышает ширину типа данных, зависит от целевой системы, как BYTE, WORD, DWORD и LWORD будут заполнены. Некоторые вызывают заполнение нулями (0), другие - `n MOD <register width>`.

Примеры

Следующий пример в шестнадцатеричной нотации показывает результаты арифметической операции в зависимости от типа входной переменной (BYTE или WORD).

Пример в ST

```
PROGRAM shr_st
VAR
  in_byte : BYTE:=16#45; (* 2#01000101 )
  in_word : WORD:=16#0045; (* 2#0000000001000101 )
  erg_byte : BYTE;
  erg_word : WORD;
  n: BYTE :=2;
END_VAR
erg_byte:=SHR(in_byte,n); (* Result is 16#11, 2#00010001 *)
erg_word:=SHR(in_word,n); (* Result is 16#0011, 2#0000000000010001 *)
```

Пример в FBD

Пример в IL

```
LD      in_byte
SHR     2
ST      erg_byte
```

ROL

Обзор

IEC оператор для поразрядного циклического сдвига операнда влево.

```
erg:= ROL (in, n)
```

Разрешённые типы данных

- BYTE
- WORD
- DWORD
- LWORD

`in` будет смещен на 1 позицию двоичного разряда влево `n` раз, в то время как бит, который является самым дальним слева, будет повторно вставлен справа.

ПРИМЕЧАНИЕ: количество битов, которое рассматривают для арифметической операции, зависит от типа данных входной переменной. Если входная переменная - константа, рассматривают самый маленький возможный тип данных. Тип данных выходной переменной не имеет никакого эффекта вообще на арифметическую операцию.

Примеры

Рассмотрите в следующем примере в шестнадцатеричной нотации различные результаты для erg_byte и erg_word. Результат зависит от типа данных входной переменной (BYTE или WORD), несмотря на то, что значения входных переменных in_byte и in_word являются теми же.

Пример в ST

```
PROGRAM rol_st
VAR
 in_byte : BYTE:=16#45;
 in_word : WORD:=16#45;
 erg_byte : BYTE;
 erg_word : WORD;
 n: BYTE :=2;
END_VAR
erg_byte:=ROL(in_byte,n); (* Result is 16#15 *)
erg_word:=ROL(in_word,n); (* Result is 16#0114 *)
```

Пример в FBD

Пример в IL

```
LD      in_byte
ROL     n
ST      erg_byte
```

ROR

Обзор

IEC оператор для поразрядного циклического сдвига операнда вправо.

`erg:= ROR (in, n)`

Разрешённые типы данных

- BYTE
- WORD
- DWORD
- LWORD

`in` будет смещен на 1 позицию двоичного разряда вправо `n` раз, в то время как бит, который является самым дальним справа, будет повторно вставлен слева.

ПРИМЕЧАНИЕ: число битов, которое рассматривают для арифметической операции, зависит от типа данных входной переменной. Если входная переменная - константа, рассматривают самый маленький возможный тип данных. Тип данных выходной переменной не имеет вообще никакого эффекта на арифметическую операцию.

Примеры

Рассмотрите в следующем примере в шестнадцатеричной нотации различные результаты для `erg_byte` и `erg_word`. Результат зависит от типа данных входной переменной (BYTE или WORD), несмотря на то, что значения входных переменных `in_byte` и `in_word` являются одинаковыми.

Пример в ST

```
PROGRAM ror_st
VAR
 in_byte : BYTE:=16#45;
 in_word : WORD:=16#45;
 erg_byte : BYTE;
 erg_word : WORD;
 n: BYTE :=2;
END_VAR
erg_byte:=ROR(in_byte,n); (* Result is 16#51 *)
erg_word:=ROR(in_word,n); (* Result is 16#4011 *)
```

Пример в FBD

Пример в IL

```
LD      in_byte
ROR     n
ST      erg_byte
```

Раздел 30.4

Операторы выбора

Обзор

Операции выбора могут также быть выполнены с переменными.

В целях ясности примеры, приведенные в этом документе, ограничены такими, которые используют константы в качестве операторов:

- SEL *(см. стр. 646)*
- MAX *(см. стр. 647)*
- MIN *(см. стр. 648)*
- LIMIT *(см. стр. 649)*
- MUX *(см. стр. 650)*

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
SEL	646
MAX	647
MIN	648
LIMIT	649
MUX	650

SEL

Обзор

IEC оператор выбора для бинарного выбора.

`G` определяет `IN0` или `IN1` присваивается выходу `OUT`.

`OUT := SEL(G, IN0, IN1)` означает:

`OUT := IN0;` если G=FALSE

`OUT := IN1;` если G=TRUE

Разрешённые типы данных:

`(IN0, IN1, OUT) :` любой тип

`G:` BOOL

Пример в IL

```
LD        TRUE
SEL       3,4                (* IN0 = 3, IN1 =4 *)
ST        Var1   (* result is 4 *) LD   FALSE
SEL       3,4
ST        Var1   (* result is 3 *)
```

Пример в ST

```
Var1:=SEL(TRUE,3,4); (* result is 4 *)
```

Пример в FBD

Примечание

ПРИМЕЧАНИЕ: выражение, расположенное перед `IN1`, не будет обработано, если `IN0` равно TRUE. Выражение, расположенное перед `IN2`, не будет обработано, если `IN0` равно FALSE.

MAX

Обзор

IEC оператор выбора, выполняющий функцию максимума.

Оператор MAX возвращает большее из 2 значений.

`OUT := MAX(IN0, IN1)`

`IN0, IN1` и `OUT` могут быть переменными любого типа.

Пример в IL

Результат равен 90

```
LD      90
MAX     30
MAX     40
MAX     77
ST      Var1
```

Пример в ST

```
Var1:=MAX(30,40); (* Result is 40 *)
Var1:=MAX(40,MAX(90,30)); (* Result is 90 *)
```

Пример в FBD

MIN

Обзор

IEC оператор выбора, выполняющий функцию минимума.

Оператор MIN возвращает меньшее из 2 значений.

```
OUT := MIN(IN0, IN1)
```

`IN0, IN1` и OUT могут быть переменной любого типа.

Пример в IL

Результат равен 30

```
LD      90
MIN     30
MIN     40
MIN     77
ST      Var1
```

Пример в ST

```
Var1:=MIN(90,30); (* Result is 30 *);
Var1:=MIN(MIN(90,30),40); (* Result is 30 *);
```

Пример в FBD

LIMIT

Обзор

IEC оператор выбора выполняет функцию ограничения.

`OUT := LIMIT(Min, IN, Max)` означает:

`OUT := MIN (MAX (IN, Min), Max)`

`Max` - верхний и `Min` - нижний предел для результата. Если значение `IN` превышает верхний предел `Max`, `LIMIT` возвращает `Max`, Если `IN` уменьшается ниже `Min`, результатом будет `Min`.

`IN` и `OUT` могут быть переменными любого типа.

Прммер в IL

Результат равен 80

```
LD      90
LIMIT   30      ,
        80
ST      Var1
```

Прммер в ST

```
Var1:=LIMIT(30,90,80); (* Result is 80 *);
```

MUX

Обзор

IEC оператор выбора для операции мультиплексирования.

`OUT := MUX(K, IN0,...,INn))` означает:

`OUT := INk`

`IN0, ...,INn` и `OUT` могут быть переменной любого типа.

`K` должно быть BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, LINT, ULINT или UDINT.

`MUX` выбирает значение `K`-ое из группы значений.

Пример в IL

Результат равен 30

```
LD      0
MUX     30      ,
        40      ,
        50      ,
        60      ,
        70      ,
        80
ST      Var1
```

Пример в ST

```
Var1:=MUX(0,30,40,50,60,70,80); (* Result is 30 *);
```

ПРИМЕЧАНИЕ: Выражение, расположенное перед другим входом кроме `INk`, не будет обработано, чтобы сохранить время выполнения. Только в режиме моделирования будут выполняться все выражения.

Раздел 30.5

Операторы сравнения

Обзор

Доступны следующие операторы, соответствующие стандарту IEC1131-3:

- GT *(см. стр. 652)*
- LT *(см. стр.653)*
- LE *(см. стр. 654)*
- GE *(см. стр. 655)*
- EQ *(см. стр.656)*
- NE *(см. стр. 657)*

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
GT	652
LT	653
LE	654
GE	655
EQ	656
NE	657

GT

Обзор

Оператор сравнения, выполняющий функцию «больше чем (Greater Than)».

Оператор GT - логический оператор, который возвращает значения TRUE, когда значение первого операнда больше, чем второго.

Операнды могут иметь любой базовый тип данных.

Пример в IL

Результат равен FALSE

```
LD      20
GT      30
ST      Var1
```

Пример в ST

```
VAR1 := 20 > 30;
```

Пример в FBD

LT

Обзор

Оператор сравнения, выполняющий функцию «меньше чем (Less Than)».

Оператор LT - логический оператор, который возвращает значениеTRUE, когда значение первого операнда меньше, чем второго.

Операнды могут иметь любой базовый тип данных.

Пример в IL

Результат равен TRUE

```
LD      20
LT      30
ST      Var1
```

Пример в ST

```
VAR1 := 20 < 30;
```

Пример в FBD

LE

Обзор

Оператор сравнения, выполняющий функцию «меньше чем или равно (Less Than Or Equal To)».

Оператор LE - логический оператор, который возвращает значение TRUE, когда значение первого операнда меньше чем или равно второму.

Операнды могут иметь любой базовый тип данных

Пример в IL

Результат равен TRUE

```
LD        20
LE        30
ST        Var1
```

Пример в ST

```
VAR1 := 20 <= 30;
```

Пример в FBD

GE

Обзор

Оператор сравнения, выполняющий функцию «больше чем или равно (Greater Than Or Equal To)».

Оператор GE - логический оператор, который возвращает значение TRUE, когда значение первого операнда больше чем или равно второму.

Операнды могут иметь любой базовый тип данных.

Пример в IL

Результат равен TRUE

```
LD      60
GE      40
ST      Var1
```

Пример в ST

```
VAR1 := 60 >= 40;
```

Пример в FBD

EQ

Обзор

Оператор сравнения, выполняющий функцию «равно (Equal To)».

The operands can be of any basic data type. Операнды могут иметь любой базовый тип данных.

Пример в IL

Результат равен TRUE

```
LD      40
EQ      40
ST      Var1
```

Пример в ST

```
VAR1 := 40 = 40;
```

Пример в FBD

NE

Обзор

Оператор сравнения, выполняющий функцию «не равно (Not Equal To)».

Оператор NE - логический оператор, который возвращает значение TRUE, когда операнды не равны.

Операнды могут иметь любой базовый тип данных.

Пример в IL

```
LD      40
NE      40
ST      Var1
```

Пример в ST

```
VAR1 := 40 <> 40;
```

Пример в FBD

Раздел 30.6

Адресные операторы

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
ADR	659
Content Operator – Оператор контента	660
BITADR	661

ADR

Обзор

Эта операция получения адреса не определена стандартом IEC 61131-3.

ADR возвращает адрес *(см. стр. 725)* ее аргумента в DWORD. Этот адрес может быть присвоен указателю *(см. стр. 587)* в проекте.

ПРИМЕЧАНИЕ: SoMachine позволяет Вам использовать оператор ADR с именами функций, именами программы, именами функциональных блоков и именами методов.

Обратитесь к главе *Указатели (см. стр. 587)*, и учтите, что указатели функций могут быть переданы внешним библиотекам. Тем не менее, нет никакой возможности вызвать указатель функции в SoMachine. Чтобы разрешить системный вызов (система выполнения), установите соответствующее свойство (в меню **View**→**Properties...**→**Build**) для объекта функции.

Пример в ST

```
dwVar:=ADR(bVAR);
```

Пример в IL

```
LD        bVar ADR
ST        dwVar
```

Соображения для онлайновых изменений

Выполнение команды **Online Change** может переместить переменные в другое место в памяти. Есть индикация во время онлайнового изменения, если необходимо копирование.

Сдвиг переменных может иметь эффект, при котором переменные указателей POINTER указывают на недопустимую память. Поэтому, гарантируйте, что указатель не сохраняется между циклами, но повторно присваивается в каждом цикле.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННОЕ ДЕЙСТВИЕ ОБОРУДОВАНИЯ

Присваивайте значение любых переменных типа указателя (POINTER TO), до их первого использования в POU, и в каждом последующем цикле.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

ПРИМЕЧАНИЕ: переменные УКАЗАТЕЛЬ НА (POINTER TO) функций и методов не должен быть возвращен к вызывающей стороне этой функции или передан глобальным переменным.

Content Operator - Оператор содержания

Обзор

Эта адресная операция не определена стандартом IEC 61131-3. Вы можете разыменовать указатель, добавив оператора содержания ^ (каре ASCII или символ циркумфлекса) после идентификатора указателя.

Пример в ST

```
pt:POINTER TO INT;
var_int1:INT;
var_int2:INT;
pt := ADR(var_int1);
var_int2:=pt^;
```

Соображения для онлайновых изменений

Выполнение команды **Online Change** может изменить содержание адресов.

⚠ ВНИМАНИЕ

НЕВЕРНЫЙ УКАЗАТЕЛЬ

При использовании указателей на адресах и выполнении команды **Online Change**, всегда проверяйте содержание указателей.

Отказ следовать этим инструкциям может привести к серьезной травме или повреждению оборудования.

BITADR

Обзор

Этот адресный оператор не определен стандартом IEC 61131-3.

`BITADR` возвращает смещение бита в сегменте в формате DWORD. Значение смещения зависит от того, активирована ли опция **Byte addressing** в целевых параметрах настройки, или нет.

Старший полубайт (нибл) этого DWORD указывает область памяти:

Memory: 16x40000000

Input: 16x80000000

Output: 16xC0000000

Пример в ST

```
VAR
      var1 AT %IX2.3:BOOL;
      bitoffset: DWORD;
END_VAR
bitoffset:=BITADR(var1); (* Result if byte addressing=TRUE: 16x80000013
, if byte addressing=FALSE: 16x80000023 *)
```

Пример в IL

```
LD        Var1 BITADR
ST        bitoffset
```

Соображения для онлайновых изменений

Выполнение команды **Online Change** может изменить содержание адресов.

⚠ ВНИМАНИЕ

НЕВЕРНЫЙ УКАЗАТЕЛЬ

При использовании указателей на адресах и выполнении команды **Online Change**, всегда проверяйте содержание указателей.

Отказ следовать этим инструкциям может привести к серьезной травме или повреждению оборудования.

Раздел 30.7

Calling Operator Оператор вызова

CAL

Обзор

IEC оператор для вызова функционального блока или программы.

Используйте `CAL` в IL для вызова экземпляра функционального блока. Поместите переменные, которые будут служить входными переменными, в круглых скобках прямо после имени экземпляра функционального блока.

Пример

Вызов экземпляра `Inst` функционального блока, где входные переменные `Par1` и `Par2` равны 0 и TRUE, соответственно.

```
CAL INST(PAR1 := 0, PAR2 := TRUE)
```

Раздел 30.8

Операторы преобразования типов

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Функции преобразования типов	664
Преобразования BOOL_TO	665
Преобразования TO_BOOL	667
Преобразование между типами целого числа	669
Преобразования REAL_TO / LREAL_TO	670
Преобразования TIME_TO/TIME_OF_DAY	672
Преобразования DATE_TO/DT_TO	674
Преобразования STRING_TO	676
`TRUNC`	678
TRUNC_INT	679
Преобразования `ANY_..._TO`	679

Функции преобразования типов

Обзор

Не разрешено неявно преобразовать от большего типа к меньшему типу (например, от INT до BYTE или от DINT до WORD). Чтобы выполнить это, Вы должны произвести специальные преобразования типов. Обычно Вы можете преобразовать любой элементарный тип к любого другому элементарному типу.

Синтаксис

```
<elem.type1>_TO_<elem.type2>
```

ПРИМЕЧАНИЕ: В преобразованиях ...TO_STRING строка генерируется, как выровненнная влево по ширине. Если она будет определена слишком короткой, она будет обрезана с правой стороны.

Поддерживаются следующие преобразования типов:

- преобразования BOOL_TO *(см. стр. 665)*
- преобразования TO_BOOL *(см. стр. 667)*
- преобразование между типами целого числа *(см. стр. 669)*
- преобразования REAL_TO-/ LREAL_TO *(см. стр. 670)*
- преобразования TIME_TO/TIME_OF_DAY *(см. стр. 672)*
- преобразования DATE_TO/DT_TO *(см. стр. 674)*
- преобразования STRING_TO *(см. стр. 676)*
- TRUNC *(см. стр. 678)* (преобразование в DINT)
- TRUNC_INT *(см. стр. 679)*
- ANY_NUM_TO_<numeric data type>
- преобразования ANY_..._TO *(см. стр. 679)*

Преобразования BOOL_TO

Определение

IEC оператор для преобразований из типа BOOL к любому другому типу.

Синтаксис

BOOL_TO_<data type>

Результаты преобразования

Результаты преобразования для типов числа и для типов строки зависят от состояния операнда:

Состояние операнда	Результат для типов числа	Результат для типов строки
TRUE	1	TRUE
FALSE	0	FALSE

Примеры в ST

Примеры в ST с результатами преобразования:

Пример	Результат
`i:=BOOL_TO_INT(TRUE);`	`1`
`str:=BOOL_TO_STRING(TRUE);`	`TRUE`
`t:=BOOL_TO_TIME(TRUE);`	`T#1ms`
`tof:=BOOL_TO_TOD(TRUE);`	`TOD#00:00:00.001`
`dat:=BOOL_TO_DATE(FALSE);`	`D#1970`
`dandt:=BOOL_TO_DT(TRUE);`	`DT#1970-01-01-00:00:01`

Примеры в IL

Примеры в IL с результатами преобразования:

Пример	Результат

`LD TRUE`
`BOOL_TO_INT`
`ST i`
	`1`

`LD TRUE`
`BOOL_TO_STRI...`
`ST str`
	`TRUE`

`LD TRUE`
`BOOL_TO_TIME`
`ST t`
	`T#1ms`

Пример	Результат
LD TRUE BOOL_TO_TOD ST tof	TOD#00:00:00.001
LD FALSE BOOL_TO_DATE ST dandt	D#1970-01-01
LD TRUE BOOL_TO_DT ST dandt	DT#1970-01-01-00:00:01

ПримерЫ в FBD

Примеры в FBD с результатами преобразования:

Пример	Результат
	1
	TRUE
	T#1ms
	TOD#00:00:00.001
	D#1970-01-01
	DT#1970-01-01-00:00:01

Преобразования TO_BOOL

Определение

IEC оператор для преобразований из переменной другого типа в BOOL.

Синтаксис

<data type>_TO_BOOL

Результаты преобразования

Результат равен TRUE, когда операнд не равен 0. Результат равен FALSE, когда операнд равен 0.

Результат равен TRUE для переменных типа STRING, когда операнд равен TRUE. Иначе результат – FALSE.

Примеры в ST

Примеры в ST с результатами преобразования:

Пример	Результат
`b := BYTE_TO_BOOL(2#11010101);`	TRUE
`b := INT_TO_BOOL(0);`	FALSE
`b := TIME_TO_BOOL(T#5ms);`	TRUE
`b := STRING_TO_BOOL('TRUE');`	TRUE

Примеры в IL

Примеры в IL с результатами преобразования

Пример	Результат

`LD 213`
`BYTE_TO_BOOL`
`ST b`
	TRUE

`LD 0`
`INT_TO_BOOL`
`ST b`
	FALSE

`LD T#5ms`
`TIME_TO_BOOL`
`ST b`
	TRUE

`LD TRUE`
`STRING_TO_BOOL`
`ST b`
	TRUE

Примеры в FBD

Примеры в FBD с результатами преобразования:

Пример	Результат
	TRUE
	FALSE
	TRUE
	TRUE

Преобразование между типами целого числа

Определение

Преобразование из числа целого типа в другое число целого типа.

Синтаксис

<INT data type>_TO_<INT data type>

Для получения информации о целочисленном типе данных обратитесь к главе *Стандартные типы данных* *(см. стр. 579)*.

Результаты преобразования

Если число, которое Вы преобразовываете, превысит предел диапазона, первые байты числа будут проигнорированы.

Пример в ST

```
si := INT_TO_SINT(4223); (* Result is 127 *)
```

Если Вы сохраните целое число 4223 (16#107f в представлении шестнадцатеричным образом) как переменную SINT, оно будет представлено как 127 (16#7f в шестнадцатеричном представлении).

Пример в IL

```
LD   4223
INT_TO_SINT
ST   si
```

Пример в FBD

Преобразования REAL_TO / LREAL_TO

Определение

Значение будет округлено вверх или вниз к ближайшему целому числу и преобразовано в новый тип переменной.

Исключения для этого - следующие типы переменных:

- STRING
- BOOL
- REAL
- LREAL

Результаты преобразования

Если REAL или LREAL будут преобразованы в SINT, USINT, INT, UINT, DINT, UDINT, LINT или ULINT, и значение вещественного числа будет находиться вне диапазона значений того целого числа, результат будет не определен, и может привести к исключению контроллера.

ПРИМЕЧАНИЕ: Проверьте любые переполнения диапазона своим приложением и проверьте, что значение REAL или LREAL находится в границах целевого целого числа, прежде чем выполнить преобразование.

При преобразовании в тип STRING обратите внимание, что общее количество цифр ограничено 16. Если у (L) REAL будет больше цифр, то шестнадцатая будет округлена. Если длина STRING будет определена слишком короткой, она будет обрезана с правого конца.

Пример в ST

Примеры в ST с результатами преобразования:

Пример	Результат
`i := REAL_TO_INT(1.5);`	2
`j := REAL_TO_INT(1.4);`	1
`i := REAL_TO_INT(-1.5);`	-2
`j := REAL_TO_INT(-1.4);`	-1

Пример в IL

```
LD                  2.75
REAL_TO_INT
ST                  i
```

Пример в FBD

Преобразования TIME_TO/TIME_OF_DAY

Определение

IEC оператор для преобразований из переменной типа TIME или TIME_OF_DAY к другому типу.

Синтаксис

TIME_TO_<data type>
TOD_TO_<data type>

Результаты преобразования

Время будет сохранено внутренне в DWORD в миллисекундах (начинающийся с 12:00 для переменной TIME_OF_DAY). Затем это значение будет преобразовано.

В случае типа STRING результатом является константа времени.

Примеры в ST

Примеры в ST с результатами преобразования:

Пример	Результат
`str := TIME_TO_STRING(T#12ms);`	`T#12ms`
`dw := TIME_TO_DWORD(T#5m);`	`300000`
`si := TOD_TO_SINT(TOD#00:00:00.012);`	`12`

Примеры в IL

Примеры в IL с результатами преобразования:

Пример	Результат

`LD T#12ms`
`TIME_TO_STRI...`
`ST str`
	`T#12ms`

`LD T#300000ms`
`TIME_TO_DWORD`
`ST dw`
	`300000`

`LD TIME_TO_SINT TOD#00:00:00.012`
`ST`
`si`
	`12`

Примеры в FBD

Преобразования DATE_TO/DT_TO

Определение

IEC оператор для преобразований переменной из типа DATE или DATE_AND_TIME в другой тип.

Синтаксис

DATE_TO_<data type>

DT_TO_<data type>

Результаты преобразования

Дата будет сохранена внутри в DWORD в секундах начиная с 1 января 1970. Затем это значение будет преобразовано.

Для переменных типа STRING результатом является константа даты.

Примеры в ST

Примеры в ST с результатами преобразования:

Пример	Результат
`b := DATE_TO_BOOL(D#1970-01-01);`	`FALSE`
`i := DATE_TO_INT(D#1970-01-15);`	`29952`
`byt := DT_TO_BYTE(DT#1970-01-15-05:05:05);`	`129`
`str := DT_TO_STRING(DT#1998-02-13-14:20);`	`DT#1998-02-13-14:20`

Примеры в IL

Примеры в IL с результатами преобразования:

Пример	Результат

`LD D#1970-01-01`
`DATE_TO_BOOL`
`ST b`
	`FALSE`

`LD D#1970-01-01`
`DATE_TO_INT`
`ST i`
	`29952`

`LD D#1970-01-15-05:05:`
`DATE_TO_BYTE`
`ST byt`
	`129`

`LD D#1998-02-13-14:20 DATE_TO_STRI...` `ST str`	`'DT#1998-02-13-14:20'`

Примеры в FBD

Преобразования STRING_TO

Определение

IEC оператор для преобразований из переменной типа STRING в другой тип.

Синтаксис

STRING_TO_<data type>

Определение значений

Определите операнд типа STRING, соответствующий стандарту IEC61131-3. Значение должно соответствовать допустимой константе (литералу) *(см. стр. 520)* целевого типа. Это применяется к спецификации экспоненциальных значений, бесконечных значений, префиксов, символ группировки (" _ ") и запятой. После цифр числа разрешены дополнительные символы, например, 23xy. Символы, предшествующие числу, не разрешены.

Операнд должен представлять допустимое значение целевого типа данных.

ПРИМЕЧАНИЕ: Если тип данных операнда не соответствует целевому типу, или если значение превышает диапазон целевого типа данных, то результат зависит от типа процессора и поэтому не определен.

Преобразования от больших типов к меньшим типам могут привести к потере информации.

⚠ ПРЕДУПРЕЖДЕНИЕ

ПОТЕРЯ ДАННЫХ

При преобразовании несоответствующих типов данных, или когда преобразовываемое значение больше, чем целевой тип данных, быть уверенным, что результат достоверен в Вашем приложении.

Отказ следовать этим инструкциям может привести к серьезной травме или повреждению оборудования.

Пример в IL

Пример	Результат преобразования

`LD 'TRUE'`
`STRING_TO_BOOL`
`ST b`
	`TRUE`

Примеры в ST

Пример	Результат преобразования
`b := STRING_TO_BOOL('TRUE');`	TRUE
`w := STRING_TO_WORD('abc34');`	0
`w := STRING_TO_WORD('34abc');`	34
`t := STRING_TO_TIME('T#127ms');`	T#127ms
`r := STRING_TO_REAL('1.234');`	1.234
`bv := STRING_TO_BYTE('500');`	244

Пример в FBD

Example Пример	Conversion result Результат преобразования
	TRUE

TRUNC

Определение

IEC оператор для преобразований из REAL в DINT. Будет использоваться целая часть значения числа.

Результат этих функций не определен, если входное значение не может быть представлено в DINT или INT. Поведение при таких входных значениях зависимо от платформы.

Примеры в ST

Примеры в ST с результатами преобразования

Пример	Результат
`diVar:=TRUNC(1.9);`	`1`
`diVar:=TRUNC(-1.4);`	`-1`

Пример в IL

```
LD          1.9
TRUNC
ST          diVar
```

TRUNC_INT

Определение

IEC оператор для преобразований из REAL в INT. Будет использоваться целая часть значения числа.

Результат этих функций не определен, если входное значение не может быть представлено в DINT или INT. Поведение при таких входных значениях зависимо от платформы.

Примеры в ST

Примеры в ST с результатами преобразования:

Пример	Результат
`iVar:=TRUNC_INT(1.9);`	1
`iVar:=TRUNC_INT(-1.4);`	-1

Пример в IL

```
LD              1.9
TRUNC_INT
ST              iVar
```

Преобразования ANY_..._TO

Определение

Преобразование из любого типа данных, или более специально, из любых числовых данных к другому типу данных. Как при любом типе преобразования, должен быть принят во внимание размер операндов, чтобы иметь успешное преобразование.

Синтаксис

ANY_NUM_TO_<numeric data type>

ANY_TO_<any data type>

Пример

Преобразование из переменной данных типа REAL в INT:

```
re : REAL := 1.234;
i : INT := ANY_TO_INT(re)
```

Раздел 30.9

Числовые функции

Обзор

В этой главе описываются доступные числовые операторы IEC, определенные стандартным IEC 61131 - 3.

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
ABS	681
SQRT	682
LN	683
LOG	684
EXP	685
SIN	686
COS	687
TAN	688
ASIN	689
ACOS	690
ATAN	691
EXPT	692

ABS

Определение

Числовой оператор IEC для возврата абсолютного значения числа.

Вход и выход могут иметь любой числовой базовый тип данных.

Пример в IL

Результат в `i` равен 2

```
LD          2
ABS
ST          i
```

Пример в ST

```
i:=ABS(-2);
```

Пример в FBD

SQRT

Определение

Числовой оператор IEC для возврата квадратного корня числа.

Входная переменная может иметь любой числовой базовый тип данных, выходная переменная должна быть типа REAL или LREAL.

Пример в IL

Результат в `q` равен 4

```
LD          16
SQRT
ST          q
```

Пример в ST

```
q:=SQRT(16);
```

Пример в FBD

LN

Определение

Числовой оператор IEC для возврата натурального логарифма числа.

Входная переменная может иметь любой числовой базовый тип данных, выходная переменная должна быть типом REAL или LREAL.

Пример в IL

Результат в q равен 3.80666

```
LD          45
LN
ST          q
```

Пример в ST

```
q:=LN(45);
```

Пример в FBD

LOG

Определение

Числовой оператор IEC для возврата логарифма числа по основанию 10 (десятичного логарифма числа).

Входная переменная может иметь любой числовой базовый тип данных, выходная переменная должна быть типом REAL или LREAL.

Пример в IL

Результат в q равен 2.49762.

```
LD          314.5
LOG
ST          q
```

Пример в ST

```
q:=LOG(314.5);
```

Пример в FBD

EXP

Определение

Числовой оператор IEC для возврата экспоненциальной функции.

Входная переменная может иметь любой числовой базовый тип данных, выходная переменная должна быть типом REAL или LREAL.

Пример в IL

Результат в `q` равен 7.389056099.

```
LD          2
EXP
ST          q
```

Пример в ST

```
q:=EXP(2);
```

Пример в FBD

SIN

Определение

Числовой оператор IEC для возврата синуса угла.

Вход, определяющий угол в радианах, может иметь любой числовой базовый тип данных, выходная переменная должна иметь тип REAL или LREAL.

Пример в IL

Результат в q равен 0.479426.

```
LD          0.5
SIN
ST          q
```

Пример в ST

```
q:=SIN(0.5);
```

Пример в FBD

COS

Определение

Числовой оператор IEC для возврата косинуса угла.

Вход, определяющий угол в радианах, может иметь любой числовой базовый тип данных, а выходная переменная должна иметь тип REAL или LREAL.

Пример в IL

Результат в `q` равен 0.877583.

```
LD          0.5
COS
ST          q
```

Пример в ST

```
q:=COS(0.5);
```

Пример в FBD

TAN

Определение

Числовой оператор IEC для возврата тангенса числа. Входное значение задаётся в радианах.

Входная переменная может иметь любой числовой базовый тип данных, а выходная переменная должна быть типом REAL или LREAL.

Пример в IL

Результат в q равен 0.546302.

```
LD          0.5
TAN
ST          q
```

Пример в ST

```
q:=TAN(0.5);
```

Пример в FBD

ASIN

Определение

Числовой оператор IEC для возврата арксинуса (обратная функция синуса) числа. Значение вычисляется в радианах.

Входная переменная может иметь любой числовой базовый тип данных, а выходная переменная должна быть типом REAL или LREAL.

Пример в IL

Результат в `q` равен 0.523599.

```
LD          0.5
ASIN
ST          q
```

Пример в ST

```
q:=ASIN(0.5);
```

Пример в FBD

ACOS

Определение

Числовой оператор IEC для возврата арккосинуса (обратная функция косинуса) числа. Значение вычисляется в радианах.

Входная переменная может иметь любой числовой базовый тип данных, а выходная переменная должна быть типом REAL или LREAL.

Пример в IL

Результат в q равен 1.0472.

```
LD   0.5
ACOS
ST   q
```

Пример в ST

```
q:=ACOS(0.5);
```

Пример в FBD

ATAN

Определение

Числовой оператор IEC для возврата арктангенса (обратная функция тангенса) числа. Значение вычисляется в радианах.

Входная переменная может иметь любой числовой базовый тип данных, а выходная переменная должна быть типом REAL или LREAL.

Пример в IL

Результат в `q` равен 0.463648.

```
LD          0.5
ATAN
ST          q
```

Пример в ST

```
q:=ATAN(0.5);
```

Пример в FBD

EXPT

Определение

Числовой оператор IEC для возведения переменной в степень другой переменной:

```
OUT = IN1 в IN2
```

Входная переменная может иметь любой числовой тип исходных данных, а выходная переменная должна быть типом REAL или LREAL.

Результат этой функции не определен, если применяется следующее:

- Основание отрицательно.
- Основание - нуль, и степень - ≤ 0.

Поведение для таких входных значений зависит от платформы.

Пример в IL

Результат равен 49.

```
LD          7
EXPT        2
ST          Var1
```

Пример в ST

```
var1:=EXPT(7,2);
```

Пример в FBD

Раздел 30.10

Операторы расширения IEC

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Операторы расширения IEC	694
__DELETE	695
__ISVALIDREF	698
__NEW	699
__QUERYINTERFACE	702
__QUERYPOINTER	704
Операторы контекста	706

Операторы расширения IEC

Обзор

В дополнение к операторам IEC, SoMachine поддерживает следующие операторы расширения IEC:

- ADR *(см. стр. 659)*
- BITADR *(см. стр. 661)*
- SIZEOF *(см. стр. 631)*
- __DELETE *(см. стр. 695)*
- __ISVALIDREF *(см. стр. 698)*
- __NEW *(см. стр. 699)*
- __QUERYINTERFACE *(см. стр. 702)*
- __QUERYPOINTER *(см. стр. 704)*
- операторы контекста (scope operators) *(см. стр. 706)*

__DELETE

Определение

Этот оператор не определен стандартом IEC 61131-3.

ПРИМЕЧАНИЕ: По причинам совместимости, версия компилятора SoMachine должна быть больше чем или равной 3.1.10.1.

Для получения дополнительной информации обратитесь к таблице версий компилятора в Руководстве пользователя по совместимости и миграции SoMachine/CoDeSys *(см. SoMachine Compatibility and Migration, User Guide)*.

Оператор `__DELETE` освобождает память, выделенную прежде для объектов через оператор `__NEW` *(см. стр. 699)*.

`__DELETE` не имеет никакого возвращаемого значения, и его операнд будет установлен в 0 после операции.

Активируйте опцию **Use dynamic memory allocation** (использование динамического выделение памяти) в виде **Application build options** (**View** → **Properties...** → **Application build options**).

Синтаксис

```
__DELETE (<pointer>)
```

Если указатель является указателем на функциональный блок, специальный метод `FB_Exit` будет вызван, прежде чем указатель будет установлен в `NULL`.

ПРИМЕЧАНИЕ: Используйте точный тип данных производного функционального блока, а не базового функционального блока. Не используйте переменную типа `POINTER TO BaseFB`. Это необходимо, потому что, если базовый функциональный блок не реализует функции `FB_Exit`, то при более позднем использовании `__DELETE(pBaseFB)`, FB_Exit не вызывается.

Пример

В следующем примере функциональный блок `FBDynamic` динамически размещается через `_NEW` из POU `PLC_PRG`. Делая так, вызывают метод `FB_Init`, в котором размещается тип `DUT`.

Когда вызывается `__DELETE` указателя функционального блока из PLC_PRG, будет вызван FB_Exit, который освобождает выделенный внутренний тип.

```
FUNCTION_BLOCK FBDynamic
VAR_INPUT
in1, in2 : INT;
END_VAR
VAR_OUTPUT
out : INT;
END_VAR
VAR
test1 : INT := 1234;
_inc : INT := 0;
_dut : POINTER TO DUT;
END_VAR
out := in1 + in2;


METHOD FB_Exit : BOOL
VAR_INPUT
bInCopyCode : BOOL;
END_VAR
__Delete(_dut);


METHOD FB_Init : BOOL
VAR_INPUT
bInitRetains : BOOL;
bInCopyCode : BOOL;
END_VAR
_dut := __NEW(DUT);


METHOD INC : INT
VAR_INPUT
END_VAR
_inc := _inc + 1;
INC := _inc;


PLC_PRG(PRG)
VAR
pFB : POINTER TO FBDynamic;
bInit: BOOL := TRUE;
bDelete: BOOL;
loc : INT;
END_VAR
IF (bInit) THEN
pFB := __NEW(FBDynamic);
```

```
bInit := FALSE;
END_IF
IF (pFB <> 0) THEN
pFB^(in1 := 1, in2 := loc, out => loc);
pFB^.INC();
END_IF
IF (bDelete) THEN
__DELETE(pFB);
END_IF
```

__ISVALIDREF

Определение

Этот оператор не определен стандартом IEC 61131-3.

Он позволяет Вам проверять указывает ли ссылка на допустимое значение.

Как использовать и пример, см. описание типа данных ссылки *(см. стр. 585)*.

_NEW

Определение

Этот оператор не прописан в стандарте IEC 61131-3.

ПРИМЕЧАНИЕ: По причинам совместимости, версия компилятора SoMachine должна быть больше чем или равной 3.1.10.1.

Для получения дополнительной информации обратитесь к таблице отображения версий компилятора в Руководстве пользователя по совместимости и миграции SoMachine/CoDeSys *(см. SoMachine Compatibility and Migration, User Guide)*.

Оператор `__NEW` выделяет память для экземпляров функционального блока или массивов стандартных типов данных. Оператор возвращает соответственно введенный указатель на объект. Если оператор не будет использоваться в присвоении, произойдет сообщение.

Активируйте опцию динамического выделения памяти **Use dynamic memory allocation** в виде **Application build options** (**View** → **Properties...** → **Application build options**), чтобы использовать оператор `__NEW`.

Если никакая память не смогла бы быть выделена, `_NEW` возвратит 0.

Для освобождения памяти используйте `__DELETE`.

Синтаксис

_NEW (<type>, [<size>]

Оператор создает новый объект указанного типа <type> и возвращает указатель на этот <type>. Инициализацию объекта вызывают после создания. Если возвращен 0, работа не была завершена успешно.

ПРИМЕЧАНИЕ: Используйте точно тип данных производного функционального блока, а не тип базового функционального блока. Не используйте переменную типа `POINTER TO BaseFB`. Это необходимо, потому что, если основной функциональный блок не реализует функцию `FB_Exit`, то при последующем использовании `__DELETE(pBaseFB)` не вызывается `FB_Exit`.

Если <type> является скаляром, опционный операнд <length>должен быть установлен дополнительно, и оператор создает массив скалярных типов с числом length.

Пример

```
pScalarType := _New(ScalarType, length);
```

ПРИМЕЧАНИЕ: Копирование кода при онлайновом изменении динамически созданных объектов не возможно.

Поэтому только функциональные блоки из библиотек (потому что они не могут изменяться) и функциональные блоки с атрибутом `enable_dynamic_creation` разрешены для оператора `__NEW`. Если функциональный блок изменяется с этим флагом так, чтобы был необходим код копии, выдаётся сообщение.

ПРИМЕЧАНИЕ: Код для выделения памяти должен быть не повторно входящим.

Семафор (`SysSemEnter`) используется, чтобы избежать попытки 2 задач выделять память одновременно. Таким образом, экстенсивное использование `__New` может произвести более высокий разброс (джиттер, дрожание).

Пример со скалярным типом:

```
TYPE DUT :
    STRUCT
        a,b,c,d,e,f : INT;
    END_STRUCT
END_TYPE
PROGRAM PLC_PRG
VAR
    pDut : POINTER TO DUT;
    bInit: BOOL := TRUE;
    bDelete: BOOL;
END_VAR
IF (bInit) THEN
    pDut := __NEW(DUT);
    bInit := FALSE;
END_IF
IF (bDelete) THEN
    __DELETE(pDut);
END_IF
```

Пример с функциональным блоком:

```
{attribute 'enable_dynamic_creation'}
FUNCTION_BLOCK FBDynamic
VAR_INPUT
    in1, in2 : INT;
END_VAR
VAR_OUTPUT
    out : INT;
END_VAR
VAR
    test1 : INT := 1234;
    _inc : INT := 0;
    _dut : POINTER TO DUT;
END_VAR
out := in1 + in2;
PROGRAM_PLC_PRG
VAR
    pFB : POINTER TO FBDynamic;
    loc : INT;
    bInit: BOOL := TRUE;
    bDelete: BOOL;
END_VAR
```

```
IF (bInit) THEN
    pFB := NEW(FBDynamic);
    bInit := FALSE;
END_IF
IF (pFB <> 0) THEN
    pFB^(in1 := 1, in2 := loc, out => loc);
    pFB^.INC();
END_IF
IF (bDelete) THEN
    __DELETE(pFB);
END_IF
```

Пример с массивом:

```
PLC_PRG(PRG)
VAR
    bInit: BOOL := TRUE;
    bDelete: BOOL;
    pArrayBytes : POINTER TO BYTE;
    test: INT;
    parr : POINTER TO BYTE;
END_VAR
IF (bInit) THEN
    pArrayBytes := NEW(BYTE, 25);
    bInit := FALSE;
END_IF
IF (pArrayBytes <> 0) THEN
    pArrayBytes[24] := 125;
    test := pArrayBytes[24];
END_IF
IF (bDelete) THEN
    __DELETE(pArrayBytes);
END_IF
```

__QUERYINTERFACE

Определение

Этот оператор не предписан стандартом IEC 61131-3.

При работе `__QUERYINTERFACE` разрешает преобразование типа ссылки интерфейса на другой. Оператор возвращает результат с типом BOOL. TRUE подразумевает, это преобразование успешно выполняется.

ПРИМЕЧАНИЕ: По причинам совместимости определение ссылки, которая будет преобразована, должно быть расширением базового интерфейса `__SYSTEM.IQueryInterface`, и версия компилятора должна быть ≥ 3.3.0.20.

Для получения дополнительной информации обратитесь к таблице отображения версий компилятора в Руководстве пользователя по совместимости и миграции SoMachine/CoDeSys *(см. SoMachine Compatibility and Migration, User Guide)*.

Синтаксис

__QUERYINTERFACE(<ITF_Source>, < ITF_Dest>)

Оператор требует для первого операнда ссылку на интерфейс или экземпляр функционального блока заданного типа и для второго операнда ссылку на интерфейс. После выполнения `__QUERYINTERFACE`, `ITF_Dest` хранит ссылку на заданный интерфейс, если объект, на который ссылаются из источника ITF, реализует интерфейс. В этом случае преобразование успешно, и результат оператора возвращает TRUE. Во всех других случаях оператор возвращает FALSE.

Предварительное условие для явного преобразования - это не только `ITF_Source`, но также и `ITF Dest` являются расширением интерфейса `__System.IQueryInterface`. Этот интерфейс обеспечивается неявно и не нуждается ни в какой библиотеке.

Пример

Пример в ST:

```
INTERFACE ItfBase EXTENDS __System.IQueryInterface
METHOD mbase : BOOL
END_METHOD
INTERFACE ItfDerived1 EXTENDS ItfBase
METHOD mderived1 : BOOL
END_METHOD
INTERFACE ItfDerived2 EXTENDS ItfBase
METHOD mderived2 : BOOL
END_METHOD
FUNCTION_BLOCK FB1 IMPLEMENTS ItfDerived1
METHOD mbase : BOOL
     mbase := TRUE;
END_METHOD
METHOD mderived1 : BOOL
```

```
     mderived1 := TRUE;
END_METHOD
END_FUNCTION_BLOCK
FUNCTION_BLOCK FB2 IMPLEMENTS ItfDerived2
METHOD mbase : BOOL
     mbase := FALSE;
END_METHOD
METHOD mderived2 : BOOL
     mderived2 := TRUE;
END_METHOD
END_FUNCTION_BLOCK
PROGRAMM POU
VAR
     inst1 : FB1;
     inst2 : FB2;
     itfbase1 : ItfBase := inst1;
     itfbase2 : ItfBase := inst2;
     itfderived1 : ItfDerived1 := 0;
     itfderived2 : ItfDerived2 := 0;
     bTest1, bTest2, xResult1, xResult2: BOOL;
END_VAR
xResult1 := __QUERYINTERFACE(itfbase1, itfderived1); // xResult = TRUE,
  itfderived1 <> 0
                             // references the instance inst1
xResult2 := __QUERYINTERFACE(itfbase1, itfderived2); // xResult = FALSE
, itfderived2 = 0
xResult3 := __QUERYINTERFACE(itfbase2, itfderived1); // xResult = FALSE
, itfderived1 = 0
xResult4 := __QUERYINTERFACE(itfbase2, itfderived2); // xResult = FALSE
, itfderived2 <> 0
                             // references the instance inst2
```

__QUERYPOINTER

Определение

Этот оператор не определен стандартом IEC 61131-3.

Во время выполнения `__QUERYPOINTER` присваивает ссылку интерфейса на нетипизованный указатель. Оператор возвращает результат с типом BOOL. TRUE подразумевает, что преобразование было успешно выполнено.

ПРИМЕЧАНИЕ: По причинам совместимости, определение намеченной ссылки интерфейса должно быть расширением основного интерфейса `__SYSTEM.IQueryInterface` и версия компилятора должна быть ≥ 3.3.0.20.

Для получения дополнительной информации обратитесь к таблице отображения версий компилятора SoMachine/CoDeSys в Руководстве пользователя по совместимости и миграции *(см. SoMachine Compatibility and Migration, User Guide)*.

Синтаксис

__QUERYPOINTER (<ITF_Source>, < Pointer_Dest>)

Для первого операнда оператор требует интерфейсной ссылки или экземпляра функционального блока намеченного типа и для второго операнда - нетипизованный указатель. После выполнения `__QUERYPOINTER`, `Pointer_Dest` содержит адрес ссылки на намеченный интерфейс. В этом случае преобразование успешно, и результат, возвращаемый оператором, – TRUE. Во всех других случаях оператор возвращает FALSE. `Pointer_Dest` нетипизован и может быть назначен на любой тип. Программист должен гарантировать фактический тип. Например, интерфейс может обеспечить метод, возвращающий тип кода.

Предварительное условие для явного преобразования - то, что `ITF_Source` - расширение интерфейса `__System.IQueryInterface`. Этот интерфейс обеспечен неявно и не нуждается ни в какой библиотеке.

Пример

```
TYPE KindOfFB
    (FB1 := 1, FB2 := 2, UNKOWN := -1);
END_TYPE
INTERFACE Itf EXTENDS __System.IQueryInterface
METHOD KindOf : KindOfFB
END_METHOD
FUNCTION_BLOCK F_BLOCK_1 IMPLEMENTS ITF
METHOD KindOf : KindOfFB
    KindOf := KindOfFB.FB1;
END_METHOD
FUNCTION_BLOCK F_BLOCK_2 IMPLEMENTS ITF
METHOD KindOf : KindOfFB
    KindOf := KindOfFB.FB2;
END_METHOD
FUNCTION CAST_TO_ANY_FB : BOOL
```

```
VAR_INPUT
    itf_in : Itf;
END_VAR
VAR_OUTPUT
    pfb_1: POINTER TO F_BLOCK_1 := 0;
    pfb_2: POINTER TO F_BLOCK_2 := 0;
END_VAR
VAR
    xResult1, xResult2 : BOOL;
END_VAR
IF itf_in <> 0
    CASE itf_in.KindOf OF
     KindOfFB.FB1:
         xResult1 := __QUERYPOINTER(itf_in, pfb_1);
       KindOfFB.FB2 THEN
         xResult2 := __QUERYPOINTER(itf_in, pfb_2);
    END_CASE
END_IF
CAST_TO_ANY_FB := xResult1 OR xResult2;
```

Операторы контекста

Определение

В расширении операторов IEC есть несколько возможностей снять неоднозначность доступа к переменным или модулям, если имя переменных или модуля используется многократно в рамках проекта.

Следующие операторы контекста могут использоваться для определения соответствующего пространства имен:

- оператор глобального контекста
- имя списка глобальных переменных
- имя перечисления
- пространство имен библиотеки

Оператор глобального контекста

Путь экземпляра, начинающийся с точки (.), открывает глобальную область видимости (пространство имен). Поэтому, если есть локальная переменная с тем же именем <varname>, что и глобальная переменная, то .<varname> относится к глобальной переменной.

Имя списка глобальной переменной

Вы можете использовать имя списка глобальных переменных как пространство имен для переменных, включенных в этот список. Таким образом можно объявить переменные с идентичными именами в различных списках глобальных переменных и, предшествуя имени переменной `<global variable list name>.`, можно получить доступ к желаемому.

Синтаксис

<global variable list name>.<variable>

Пример

Списки глобальных переменных `globlist1` и `globlist2`, каждый содержит переменную, названную `varx`. В следующей строке, `varx` из `globlist2` скопирован в `varx` в `globlist1`:

```
globlist1.varx := globlist2.varx;
```

Если на имя переменной, объявленное больше чем в одном списке глобальных переменных, сошлются без имени списка глобальной переменной как предшествующего оператора, будет сгенерировано сообщение.

Пространство имен библиотеки

Вы можете использовать пространство имен библиотеки, чтобы получить доступ к компонентам библиотеки явно.

Пример

Если библиотека, которая включена в проект, содержит модуль `fun1` и есть также POU `fun1`, определенный локально в проекте, то Вы можете добавить имя пространства библиотеки `namespace` к имени модуля, чтобы сделать доступ уникальным.

Синтаксис

`<namespace>.<module name>`, например `lib1.fun1`.

По умолчанию `namespace` библиотеки идентично имени библиотеки. Однако, Вы можете определить любое другое в **Project Information** при создании библиотеки проекта в **Project Information** (по умолчанию в меню **Project**), или позже в диалоговом окне **Properties** включенной библиотеки в **LibraryManager**.

Пример

Есть функция `fun1` в библиотеки `lib`. Есть также функция `fun1`, объявленная в проекте. По умолчанию пространство имен библиотеки `lib` называют `'lib'`:

```
res1 := fun(in := 12); // call of the project function fun
res2 := lib.fun(in := 12); // call of the library function fun
```

Имя перечисления

Вы можете использовать имя типа перечисления, чтобы снять неоднозначность доступа к перечислению констант. Поэтому можно использовать ту же константу в различных перечислениях.

Имя перечисления должно предшествовать имени константы, отделенному точкой (.).

Синтаксис

<enumeration name>.<constant name>

Пример

Постоянный `Blue` - компонент перечисления `Colors`, а также перечисления `Feelings`.

```
color := Colors.Blue; // Access to enum value Blue in type Colors
feeling := Feelings.Blue; // Access to enum value Blue in type Feelings
```

Раздел 30.11

Оператор инициализации

Оператор INI

Обзор

ПРИМЕЧАНИЕ: оператор `INI` устарел. Метод `FB_init` заменяет оператора `INI`. Для получения дополнительной информации о методе `FB_init`, обращайтесь к разделу FB_init, Методы FB_reinit (*см. стр. 525*). Однако оператор все еще может использоваться для сохранения совместимости с проектами, импортированными из более ранних версий SoMachine.

Вы можете использовать оператор `INI`, чтобы инициализировать сохраняемые переменные, которые обеспечиваются экземпляром функционального блока, используемым в POU.

Присваивайте оператор логической переменной.

Синтаксис

<bool-variable> := INI(<FB-instance, TRUE|FALSE)

Если второй параметр оператора установлен на TRUE, все сохраненные переменные, определенные в функциональном блоке `FB`, будут инициализированы.

Пример в ST

`fbinst` - экземпляр функционального блока `fb`, в котором определена сохраняемая переменная `retvar`.

Объявление в POU

```
fbinst:fb;
b:bool;
```

Часть реализации

```
b := INI(fbinst, TRUE);
ivar:=fbinst.retvar (* => retvar gets initialized *)
```

Пример вызова оператора в FBD

Глава 31

Операнды

Содержание главы

Эта глава содержит следующие разделы:

Раздел	Тема	Страница
31.1	Константы	710
31.2	Переменные	721
31.3	Адреса	725
31.4	Функции	728

Раздел 31.1

Константы

Содержание этого раздела

Это раздел содержит следующие темы:

Тема	Страница
Константы BOOL	711
Константы TIME	712
Константы DATE	714
Константы DATE_AND_TIME	715
Константы TIME_OF_DAY	716
Константы чисел	717
Константы REAL/LREAL	718
Константы STRING	719
Типизированные константы / Типизированные Литералы	720

Константы BOOL

Обзор

Константы BOOL - логические значения TRUE и FALSE (ИСТИНА И ЛОЖЬ). Обращайтесь к описанию типа данных BOOL (*см. стр. 579*).

Константы TIME

Обзор

Константы TIME используются, чтобы управлять стандартными модулями таймера. Время константы TIME имеет размер 32 бита и соответствует стандарту IEC 61131-3. Кроме того, LTIME поддерживается как расширение стандарта как основа времени для таймеров с высоким разрешением. LTIME имеет размер разрешения наносекунды и 64 бита.

Синтаксис для константы TIME

t#<time declaration>

Вместо t# Вы можете также использовать следующее:

- `T#`
- `time`
- `TIME`

Объявление времени может включать следующие единицы измерения времени. Они должны использоваться в следующей последовательности, но она не требуется, чтобы использовать их все.

- d: days дни
- h: hours часы
- m: minutes минуты
- s: seconds секунды
- ms: milliseconds миллисекунды

Примеры корректных констант TIME в ST присвоении

Пример	Описание
`TIME1 := T#14ms;`	–
`TIME1 := T#100S12ms;`	(* самому высокому компоненту может быть позволено превысить свой предел *),
`TIME1 := t#12h34m15s;`	–

Примеры неправильного использования

Пример	Описание
`TIME1 := t#5m68s;`	(* предел превышен в более низком компоненте *),
`TIME1 := 15ms;`	(* T# отсутствует *),
`TIME1 := t#4ms13d;`	(* неправильный порядок записей *)

Синтаксис для константы LTIME

LTIME#<time declaration>

Объявление времени может включать единицы измерения времени, как используется с константой TIME и дополнительно:

- us: микросекунды
- ns: наносекунды

Примеры корректных констант LTIME в ST присвоении:

```
LTIME1 := LTIME#1000d15h23m12s34ms2us44ns
LTIME1 := LTIME#3445343m3424732874823ns
```

Для получения дополнительной информации обратитесь к описанию типов данных TIME (*см. стр. 581*).

Константы DATE

Обзор

Используйте эти константы, чтобы ввести даты.

Синтаксис

d#<date declaration>

Вместо d# Вы можете также использовать следующее:

- `D#`
- `date`
- `DATE`

Введите объявление даты в формат <year-month-day> (<год-месяц-день>).

Значения DATE внутренне обрабатываются как DWORD значения, содержа длину отрезка времени в секундах с 01.01.1970, 0:00 часов.

Примеры

```
DATE#1996-05-06
d#1972-03-29
```

Для получения дополнительной информации обратитесь к описанию типов данных TIME (*см. стр. 581*).

Константы DATE_AND_TIME

Обзор

Константы DATE и константы TIME_OF_DAY могут быть объединены, чтобы сформировать так называемые константы DATE_AND_TIME.

Синтаксис

`dt#`<date and time declaration>

Вместо `dt#` Вы можете использовать следующее:

- `DT#`
- `date_and_time`
- `DATE_AND_TIME`

Вводите объявление даты и времени в формате <year-month-day-hour:minute:second>.

Вы можете вводить секунды как вещественные числа. Это позволяет Вам определять части секунды.

Значения DATE_AND_TIME внутренне обрабатываются как DWORD, содержа длину отрезка времени в секундах с 01.01.1970, 0:00 часов.

Примеры

```
DATE_AND_TIME#1996-05-06-15:36:30
dt#1972-03-29-00:00:00
```

Для получения дополнительной информации обратитесь к описанию типов данных TIME (*см. стр. 581*).

Константы TIME_OF_DAY

Обзор

Используйте этот тип постоянных, чтобы хранить время дня.

Синтаксис

tod#<time declaration>

Вместо `tod#` Вы можете использовать следующее:

- `TOD#`
- `time_of_day#`
- `TIME_OF_DAY#`

Вводите декларации времени в формате <hour:minute:second>.

Вы можете вводить секунды как вещественные числа. Это позволяет Вам определять части секунды.

TIME_OF_DAY внутренне обрабатываются как DWORD, содержа длину отрезка времени в секундах с 00:00 часов.

Примеры

```
TIME_OF_DAY#15:36:30.123
tod#00:00:00
```

Для получения дополнительной информации обратитесь к описанию типов данных TIME (*см. стр. 581*).

Константы чисел

Обзор

Числовые значения могут появляться как двоичные числа, восьмеричные числа, десятичные числа и шестнадцатеричные числа. Целочисленные значения, которые не являются десятичными числами, представлены основой, сопровождаемой знаком номера (#) перед целочисленной константой. Значения для номеров 10... 15 в шестнадцатеричных числах представлены буквами A... F.

Вы можете включать символ подчеркивания в числе.

Примеры

`14`	(десятичное число)
`2#1001_0011`	(двоичное число)
`8#67`	(восьмеричное число)
`16#A`	(шестнадцатеричное число)

Эти числовые значения могут быть типа:

- BYTE
- WORD
- DWORD
- SINT
- USINT
- INT
- UINT
- DINT
- UDINT
- REAL
- LREAL

Неявные преобразования от большего до меньших типов переменных не разрешены. Это означает, что переменная DINT не может просто использоваться в качестве переменной INT. Используйте функции преобразования типов *(см. стр. 664)*.

Константы REAL/LREAL

Обзор

Константы REAL и LREAL могут быть даны как десятичные дроби и представлены в виде действительного числа. Используйте для этого стандартный американский формат с десятичной точкой.

ПРИМЕЧАНИЕ: Когда Вы создаете функциональный блок с реализацией функции, которая включает константу LREAL (такую как `IF_TouchProbe`), затем вставьте мониторинг атрибута (*см. стр. 556*).

Пример:

```
{attribute 'monitoring' := 'call'}
PROPERTY CaptureValue : LREAL
```

Примеры

`7.4`	вместо `7,4`
`1.64e+009`	вместо `1,64e+009`

Константы STRING

Обзор

Строка - произвольная последовательность символов. Константы STRING (*см. стр. 581*) предшествуют и завершают одиночные кавычки. Вы можете также вводить пробелы и специальные символы (специальные символы для различных языков, как диакритические знаки или умляуты). Их будут рассматривать точно так же, как все другие символы.

В строках комбинация знака доллара ($), сопровождаемая 2 шестнадцатеричными числами, будет интерпретироваться как шестнадцатеричное представление 8-разрядного символьного кода.

Далее, есть некоторые комбинации символов, запускающихся со знака доллара, которые интерпретируются следующим образом:

Введенная комбинация	Интерпретация
`$`<два шестнадцатиричных числа>	шестнадцатеричное представление 8-разрядного символьного кода
`$$`	знак доллара
`$'`	одиночная кавычка
`$L` или `$l`	перевод строки
`$N` или `$n`	новая строка
`$P` или `$p`	перевод страницы
`$R` или `$r`	разрыв строки
`$T` или `$t`	табуляция

Примеры

```
'w1Wüß?'
' Abby and Craig '
':-)'
'costs ($$)'
```

Типизированные константы / Типизированные Литералы

Обзор

В основном, при использовании констант IEC будет использоваться самый маленький тип данных. Исключение - константы REAL/LREAL, где всегда используется LREAL. Если должен использоваться другой тип данных, используйте введенные литералы (типизированные константы) без необходимости явного объявления констант. С этой целью константе будет предоставен префикс, который определяет тип.

Синтаксис

```
<Type>#<Literal>
```

`<Type>` - желаемый тип данных. Возможные записи:

- BOOL
- SINT
- USINT
- BYTE
- INT
- UINT
- WORD
- DINT
- UDINT
- DWORD
- REAL
- LREAL

Записывайте тип прописными буквами.

`<Literal>` определяет константу. Введенные данные должны соответствовать типу данных, определенному в `<Type>`.

Пример

```
var1:=DINT#34;
```

Если константа не может быть преобразована в целевой тип без потери данных, будет сгенерировано сообщение.

Вы можете использовать введенные литералы везде, где могут использоваться нормальные константы.

Раздел 31.2

Переменные

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Переменные	722
Адресация битов в переменных	723

Переменные

Обзор

Вы можете объявить переменные или локально в разделе описаний POU, или в списке глобальных переменных (GVL - global variable list), или в списке персистентных переменных, или в распределении Вх/Вых устройств.

Обратитесь к главе *Декларация переменных* (*см. стр. 495*) для получения информации об объявлении переменной, включая правила относительно идентификатора переменной и многократного использования.

Это зависит от типа данных *(см. стр. 577)*, где может использоваться переменная.

Вы можете получить доступ к доступным переменным через **Input Assistant**.

Получение доступа к переменным для массивов, структур и POU

Таблица приводит соответствующий синтаксис для доступа к массивам, структурам и POU:

Синтаксис	Доступ к
<array name>[Index1, Index2]	компоненты 2-мерного массива ***(см. стр. 592)***
<structure name>.<variable name>	переменные структуры ***(см. стр. 595)***
<function block name>.<variable name>	функциональный блок и переменные программы

Адресация битов в переменных

Обзор

В целочисленных переменных можно получить доступ к отдельным битам. С этой целью добавьте индекс бита, который будет адресоваться к переменной и отделите его точкой. Вы можете дать любую константу разрядному индексу. Индексация начинается с 0.

Синтаксис

<variablename>.<bitindex>

Пример

```
a : INT;
b : BOOL;
...
a.2 := b;
```

Третий бит переменной `a` будет установленным на значение переменной `b`, это означает, что эта переменная `a` будет равняться 3.

Если индекс будет больше, чем разрядная ширина переменной будет сгенерировано, следующее сообщение: 'Index '<n>' outside the valid range for variable (вне допустимого диапазона для переменной) '<var>'!'

Побитовая адресация возможна с переменными следующих типов данных:

- SINT
- INT
- DINT
- USINT
- UINT
- UDINT
- BYTE
- WORD
- DWORD

Если тип данных не разрешает побитовый доступ, будет сгенерировано следующее сообщение: 'Invalid data type (Недопустимый тип данны '<type>' for direct indexing (для прямой индексации)'.

Не присваивайте побитовый доступ к переменной `VAR_IN_OUT`.

Битовый доступ через глобальную константу

Если Если Вы объявили глобальную константу, определяющую битовый индекса, Вы можете использовать эту константу для битового доступа.

Пример битового доступа через глобальную константу и к переменной:

1. Объявление глобальной константы в списке глобальных переменных

Переменная `enable` определяет бит, к которому получают доступ:

```
VAR_GLOBAL CONSTANT
enable:int:=2;
END_VAR
```

2. Битовый доступ к целочисленной переменной

Объявление в POU:

```
VAR
xxx:int;
END_VAR
```

Битовый доступ к типам данных BIT

Тип данных BIT - специальный тип данных, который позволен только в структурах. Для получения дополнительной информации обратитесь к Битовому доступу в Структурах *(см. стр. 596)*.

Пример: Битовый доступ к типах данных BIT

Объявление структуры

```
TYPE ControllerData :
STRUCT
   Status_OperationEnabled : BIT;
   Status_SwitchOnActive : BIT;
   Status_EnableOperation : BIT;
   Status_Error : BIT;
   Status_VoltageEnabled : BIT;
   Status_QuickStop : BIT;
   Status_SwitchOnLocked : BIT;
   Status_Warning : BIT;
END_STRUCT
END_TYPE
```

Объявление в POU

```
VAR
ControllerDrive1:ControllerData;
END_VAR
```

Битовый доступ

```
ControllerDrive1.OperationEnabled := TRUE;
```

Раздел 31.3

Адреса

Адрес

Соображения для онлайновых изменений

Выполнение команды **Online Change** может изменить содержание адресов.

⚠ ВНИМАНИЕ

НЕВЕРНЫЙ УКАЗАТЕЛЬ

При использовании указателей на адресах и выполнении команды **Online Change**, всегда проверяйте содержание указателей.

Отказ следовать этим инструкциям может привести к серьезной травме или повреждению оборудования.

Обзор

При определении адреса размещение памяти и размер обозначены специальными символьными последовательностями.

Синтаксис

%<memory area prefix><size prefix><number|.number|.number....>

Поддерживаются следующие префиксы области памяти:

I	вход (физические входы через входной драйвер, датчики)
Q	выход (физические выходы через выходной драйвер, исполнительные устройства)
M	ячейка памяти

Поддерживаются следующие префиксы размера:

X	одиночный бит
нет	одиночный бит
B	байт (8 бит)
W	(16 бит)
D	двойное слово (32 бита)

Примеры

Пример адреса	Описание
%QX7.5	бит выхода 7.5
%Q7.5	
%IW215	входное слово 215
%QB7	байт выхода 7
%MD48	двойное слово в позиции памяти 48 в расположении памяти
%IW2.5.7.1	интерпретация зависит от текущей конфигурации контроллера (см. ниже)
ivar AT %IW0: WORD;	пример объявления переменной, включая присвоение адреса Для получения дополнительной информации обратитесь к разделу Декларация AT *(см. стр. 508)*.

Присвоение допустимых адресов

Для присвоения допустимого адреса в приложении определите следующее:

- надлежащую позицию в изображении процесса, то есть область памяти, которая будет использоваться: I = ввод, Q = вывод или M = область памяти, как обозначено в приведенной выше таблице
- желаемый размер: X = бит, B = байт, W = слово, D = dword, как обозначено в таблице примеров

Текущая конфигурация устройства и параметры настройки (аппаратная структура, описание устройства, параметры настройки Вх/Вых) играют решающую роль. Особенно, рассмотрите различия в разрядной интерпретации адреса между устройствами, использующими байтовый способ адресации или тех, которые используют ориентированный на слово способ адресации IEC. Например, в устройстве с байтовой адресацией первый элемент битового адреса %IX5.5 будет байт с адресом 5, но в устройстве с адресацией по словам это будет слово с адресом 5. В отличие от этого, адресация слова или байта независима от типа устройства: %IW5 всегда будет адресовать слово 5, и с адресом байта %IB5 всегда байт 5.

В зависимости от размера и способа адресации, различные элементы памяти могут адресоваться одним и тем же определением адреса.

Различия между байтовой адресацией и адресацией IEC, ориентированной на слова

Смотрите приведенную ниже таблицу для сравнения байтовой адресации и ориентированной на слова IEC адресации для битов, байтов, слов и двойных слов. Она визуализирует перекрывающиеся области памяти в случае байтового способа адресации (см. пример в таблице ниже).

Относительно нотации, полагайте, что для битовых адресов способ адресации IEC всегда ориентируется на слово. Это означает, что место перед точкой соответствует количеству слов, места позади - числу битов.

Сравнение байтовой и ориентированной на слова адресаций для размеров адресов D, W, B и X:

DWords / Words				**Bytes**	**X (Bits)**					
байтовая адресация		ориентированная на слово IEC адресация			битовая адресация			ориентированная на слово IEC адресация		
D0	W0	D0	W0	B0	X0.7	...	X0.0	X0.7	...	X0.0
D1	W1	–	–	B1	X1.7	...	X1.0	X0.15	...	X0.8
...	W2	–	W1	B2	...	–	–	X1.7	...	X1.0
–	W3	–	–	B3	–	–	–	X1.15	...	X1.8
–	W4	D1	W2	B4	–	–	–	–	–	–
–	...	–	–	B5	–	–	–	–	–	–
–	–	–	W3	B6	–	–	–	–	–	–
–	–	–	–	B7	–	–	–	–	–	–
–	–	D2	W4	B8	–	–	–	–	–	–
–	–	...	–	...	–	–	–	–	–	–
–	–	...	–	...	–	–	–	–	–	–
–	–	...	–	...	–	–	–	–	–	–
D(n-3)	–	D(n/4)	...	–	–	–	–	–	–	–
–	–	–	–	–	–	–	–	–	–	–
–	W(n-1)	–	W(n/2)	–	–	–	–	–	–	–
–	–	–	–	Bn	Xn.7	...	Xn.0	X(n/2).15	...	X(n/2).8
n = номер байта										

Пример для наложения диапазонов памяти в случае байтового способа адресации:

- D0 содержит B0...B3
- W0 содержит B0 и B1
- W1 содержит B1 и B2
- W2 содержит B2 и B3

Чтобы обойти перекрытие, не используйте W1 или D1, D2, D3 для адресации

ПРИМЕЧАНИЕ: Булевы значения будут размещены побайтно, если не будет определен никакой явный однобитовый адрес. Пример: изменение в значении `varbool1 AT %QW0` влияет на диапазон от QX0.0...QX0.7.

Раздел 31.4

Функции

Функции

Обзор

В ST вызов функции может использоваться в качестве операнда.

Пример

```
Result := Fct(7) + 3;
```

Функция TIME()

Эта функция возвращает время (на основе миллисекунд), которое прошло с момента запуска системы. Типом данных является TIME.

Пример в IL

```
TIME
ST systime       (* Result for example: T#35m11s342ms *)
```

Пример в ST

```
systime:=TIME();
```

Часть VIII

Шаблоны SoMachine

Содержание этой части

Эта часть содержит следующие главы:

Глава	Название главы	Страница
32	Общая информация о шаблонах SoMachine	731
33	Управление шаблонами устройств	743
34	Управление шаблонами функций	757

Глава 32

Общая информация о шаблонах SoMachine

Раздел 32.1

Шаблоны SoMachine

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Общая информация о шаблонах SoMachine	733
Администрирование шаблонов SoMachine	736

Общая информация о шаблонах SoMachine

Обзор

SoMachine обеспечивает шаблоны, чтобы сделать специальную функциональность управления и визуализации, которая была разработана в одном проекте SoMachine, легко доступной другим проектам SoMachine. Они помогают стандартизировать использование полевых устройств и прикладных функций всюду в различных проектах SoMachine.

Доступны следующие типы шаблонов:

- Шаблоны устройства, которые связаны с единственным полевым устройством или модулем ввода-вывода
- Шаблоны функций, которые связаны с высокоуровневой прикладной функцией

SoMachine обеспечивает различные шаблоны, но Вы можете также создать свои собственные шаблоны для любой функциональности, которую Вы хотите сделать доступной для других проектов.

Создание собственных шаблонов

Следующие шаги требуются для всех шаблонов SoMachine:

Шаг	Действие
1	Создайте свою функциональность в проекте SoMachine и протестируйте ее надлежащими аппаратными средствами или моделированием.
2	Сохраните функциональность в библиотеке шаблонов.
3	Откройте другой проект SoMachine и выберите шаблон из библиотеки шаблонов, чтобы сделать функциональность доступной для этого проекта.

Общие замечания

При использовании шаблонов SoMachine отметьте следующее:

- Шаблоны не специфичны для типа контроллера и поэтому могут быть сделаны доступными для любого контроллера. Проверьте, что контроллер, к которому Вы добавляете шаблон, способен к выполнению функциональности, содержавшейся в шаблоне.
- После установки шаблона Вы можете свободно адаптировать создаваемые объекты к своим индивидуальным требованиям.
- Функция шаблонов не поддерживает приложения Vijeo-Designer; HMI приложения не включены в шаблоны SoMachine.
- Можно установить один шаблон несколько раз на том же устройстве контроллера. Во избежание конфликтов имен при создании тех же объектов несколько раз, они переименовываются автоматически во время установки. Для получения дополнительной информации обратитесь к разделу *Наименование объектов* из главы *Добавление устройств из шаблонов* (*см. стр. 748*).
- Определяемые пользователем типы данных (DUT) или функциональные блоки должны быть определены в библиотеке функциональных блоков, если они должны использоваться в шаблонах.
- Шаблоны не поддерживают использование прямых представлений переменных (например, `%IX2.0`).

Но Вы можете, с другой стороны, использовать прямые представления с неполной спецификацией адреса (например, `%I*`). Для получения дополнительной информации обратитесь к разделу *Variables configuration - VAR_CONFIG (Конфигурация переменных - VAR_CONFIG)* (*см. стр. 522*).

ПРИМЕЧАНИЕ: Несмотря на то, что эта форма заполнения доступна для прямых адресов, избегайте непосредственной адресации в своих программах и используйте символьную адресацию везде, где и каждый раз, когда возможно.

SoMachine позволяет Вам программировать инструкции, используя или прямой или косвенный метод использования параметра. Прямой метод называют непосредственной адресацией (Immediate addressing), где Вы используете прямой адрес параметра, например, такой как %IWx или %QWx. Косвенный метод называют символьной адресацией(Symbolic Addressing), где Вы сначала определяете символы для тех же параметров, и затем используете символы в объединении с инструкциями Вашей программы.

Оба метода - допустимые и приемлемые, но символьная адресация предлагают явные преимущества, особенно если Вы позже делаете модификации к своей конфигурации. Когда Вы конфигурируете Вх/Вых и другие устройства для Вашего приложения, SoMachine автоматически выделяет и присваивает прямые адреса. Позже, если Вы добавите или удалите Вх/Вых или другие устройства из Вашей конфигурации, SoMachine будет составлять любые изменения в конфигурации, перераспределяя и повторно присваивая прямые адреса. Это обязательно изменит присвоения от тех, которыми они когда-то были от точки изменения (изменений) в конфигурации.

Если Вы уже создали все или часть Вашей программы, используя прямые адреса, Вы должны будете принять во внимание эти изменение в любых инструкциях программы, функциональных блоках, и т.д., изменяя все прямые адреса, которые были переприсвоены. Однако, если Вы используете символы вместо прямых адресов в Вашей программе, это действие ненужно. Символы автоматически обновляются их новыми ассоциациями с прямым адресом при условии, что они присоединены к адресу в диалоговом окне I/O Mapping соответствующего редактора устройств (Device Editor), и не просто ''AT' объявлением в самой программе.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННОЕ ДЕЙСТВИЕ ОБОРУДОВАНИЯ

Осмотрите и измените по мере необходимости любые непосредственные адреса Вх/Вых, используемые в приложении после изменения конфигурации.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

ПРИМЕЧАНИЕ: Систематически используйте символы при программировании, чтобы помочь избежать обширных модификаций программы и ограничить возможность аномалий программирования, как только конфигурация программы была изменена, добавлением или удалением Вх/Вых или других устройств.

Поддерживаемые модули Вх/Вых

Шаблоны SoMachine могут включать следующие модули Вх/Вых:

- TM2
- TM3
- TM5

Поддерживаемые полевые шины

Шаблоны SoMachine могут включать полевые устройства, которые соединены со следующими полевыми шинами:

- CANopen
- Modbus serial line (Modbus IOScanner)
- Modbus TCP IO Scanner
- SoftMotion General Drive Pool (LMC058)
- CANmotion

Администрирование шаблонов SoMachine

Обзор

Следующие параграфы содержат обзор того, как создать новые или изменить существующие шаблоны устройств или функций и сохранить их как файлы для передачи их к другим PC.

Библиотеки шаблонов

Библиотеки шаблонов содержат определение нескольких шаблонов устройств или функций.

Защита от записи

Стандартные библиотеки шаблонов, включенные в предоставляемый объем SoMachine, защищены от записи, что означает, что они не могут быть удалены или переименованы.

ПРИМЕЧАНИЕ: Вы не можете изменить защищенные от записи библиотеки (удалением индивидуальных шаблонов или изменением имён), но Вы можете полностью удалить их.

Администрирование шаблонов

Для администрирования доступных шаблонов устройств и функций в SoMachine, выберите **Tools** → **Template Repository** в SoMachine Logic Builder. Чтобы получить доступ к **Template Repository** из SoMachine Central, откройте диалоговое окно **System Options**, нажав кнопку **System Options** на панели инструментов или нажав кнопку **Settings** в экране **Versions**. В диалоговом окне **System Options** нажмите кнопку **Template Repository** *(см. SoMachine Central, User Guide)*.

Открывается диалоговое окно **Template Repository**:

Из списка **Location** (Расположение) выберите тип шаблонов, которые будут отображены в поле **Installed Templates**:

- **<All locations>** выбрано по умолчанию: отображаются все шаблоны доступных устройств и функций
- **Legacy** отображает шаблоны устройств и функций SoMachine V3.1 (если установлены)
- **User**: отображает только те шаблоны устройств и функций, которые Вы создали или установили
- **System**: отображает шаблоны стандартных устройств и функций, предоставленные SoMachine

Путь к каталогу, где сохранены библиотеки шаблонов, отображается ниже поля **Location**.

Поле **Installed Templates** перечисляет установленные шаблоны в 2 группах: **Device Templates** и **Function Templates**. Каждая библиотека шаблонов может содержать или шаблоны устройства или шаблоны функций.

Установка дополнительных библиотек шаблонов

Чтобы добавить дополнительные библиотеки шаблонов к этому списку, выполните следующее :

Шаг	Действие
1	Нажмите кнопку **Install** в диалоговом окне **Template Repository.** **Результат:** Открывается диалоговое окно **File open**.
2	Просмотрите папку, где сохранен файл библиотеки шаблонов, который Вы хотите установить,.
3	Выберите файл библиотеки, который Вы хотите установить и нажмите "**OK**". **Результат**: выбранная библиотека шаблонов установлена и обозначена в диалоговом окне **Template Repository**, включая шаблоны устройств или функций, которые она содержит.

Удаление библиотек шаблонов

Шаг	Действие
1	В списке **Installed Templates** диалогового окна **Template Repository** выберите библиотеку шаблонов, которую Вы хотите удалить.
2	Чтобы удалить выбранную библиотеку шаблонов, нажмите кнопку **Uninstall**. **Результат**: выбранная библиотека шаблонов удалена из установки.

Переименование библиотек шаблонов

Чтобы переименовать библиотеку шаблонов, выполните следующее:

Шаг	Действие
1	В списке **Installed Templates** диалогового окна **Template Repository** выберите библиотеку шаблонов, которую Вы хотите переименовать.
2	Щелкните по имени библиотеки шаблонов, которую Вы хотите изменить. **Результат**: поле открывается.
3	Введите новое имя в поле и нажмите **Enter** или оставьте поле. **Результат**: библиотеке шаблонов теперь присвоено новое имя.

Создание новой библиотеки шаблонов

Чтобы создать новую библиотеку шаблонов, выполните следующее:

Шаг	Действие
1	Чтобы создать новую библиотеку шаблонов, выберите опцию **User** или **<All locations>** из списка **Location.**
2	Чтобы создать новую библиотеку шаблонов для шаблонов устройств, выберите узел **Device Templates** в списке **Installed Templates**, и нажмите кнопку **Create Library**. **Результат**: новая библиотека шаблонов с именем по умолчанию добавлена в нижнюю часть раздела **Device Templates** списка **Installed Templates**. Чтобы создать новую библиотеку шаблонов для шаблонов функций, выберите узел **Function Templates** в списке **Installed Templates**, и нажмите кнопку **Create Library**. **Результат**: новая библиотека шаблонов с именем по умолчанию добавлена в нижнюю часть раздела **Function Templates** списка **Installed Templates**.
3	Переименуйте новую библиотеку шаблонов как указано выше и заполните ее шаблонами устройств или функций при помощи, например, копирования и операций вставки, описанных ниже.

Сохранение библиотек шаблонов как файла

Библиотеки шаблонов, которые содержат шаблоны устройств или функций, являются SoMachine-специфичными XML-файлами. Чтобы обеспечить их использование на других PC, выполните следующее:

Шаг	Действие
1	Выберите библиотеку шаблонов, которую Вы хотите экспортировать в списке **Installed Templates**.
2	Нажмите кнопку **Save As File... .**
3	В диалоговом окне **Save File** переместитесь к папке, где Вы хотите сохранить файл библиотеки шаблонов.
4	Переместите файл библиотеки шаблонов в другой PC и установите его при помощи **Template Repository**.

Операции копирования и вставки для библиотек шаблонов

Диалоговое окно **Template Repository** также поддерживает операцию копирования и вставки для библиотек шаблонов.

Чтобы скопировать библиотеку шаблонов с шаблоном устройства или функции, который она содержит, выберите соответствующий элемент в списке **Installed Templates**, и нажимите кнопку **Copy**.

Теперь выберите **Device Templates** или узел **Function Templates**, и нажмите кнопку **Paste**, чтобы вставить копию этой библиотеки шаблонов с именем по умолчанию в списке **Installed Templates**.

Замените имя по умолчанию именем по Вашему выбору.

Операции копирования и вставки для шаблонов

Диалоговое окно **Template Repository** поддерживает операцию копирования и вставки для шаблонов устройства или функций.

Чтобы скопировать шаблон устройства или функции, выберите соответствующий элемент ниже узла библиотеки шаблонов в списке **Installed Templates** и нажмите кнопку **Copy**.

Вы можете теперь вставить шаблон в библиотеку шаблонов, если библиотека не защищена от записи. Библиотека может быть вставлена только в библиотеку того же вида.

Если Вы желаете, замените имя по умолчанию именем по Вашему выбору.

Добавление дополнительной информации для шаблонов или библиотек шаблонов

Диалоговое окно **Template Repository** позволяет Вам вводить дополнительную информацию для шаблонов или библиотек шаблонов.

Чтобы добавить информацию, выберите библиотеку, или библиотеку шаблонов в списке **Installed Templates** и нажмите кнопку **Properties...** .

Диалоговое окно **Properties** для выбранной библиотеки или библиотеки шаблонов отображено ниже.

Если выбранная библиотека или библиотека шаблонов не защищены от записи, диалоговое окно **Properties** содержит следующие параметры, которые Вы можете отредактировать, вместе с соответствующими им кнопками:

Элемент		Описание
Поле **Template Name / Library Name**		Указывает имя библиотеки или библиотеки шаблонов, к которой применяются эти свойства. Чтобы изменить имя, щелкните по этому полю и измените имя согласно Вашим требованиям.
Поле **Help ID**		Для шаблонов или библиотек шаблонов Schneider Electric содержит ссылку на соответствующее описание в онлайн-справке. Если есть документ онлайн-справки, доступный для Ваших собственных шаблонов, Вы можете ввести полную ссылку на ее расположение в онлайн-справке или ключевое слово, соответствующее индексу в онлайн-справке.
Кнопка **Show Help**		Открывает документ онлайн-справки, определенный в поле **Help ID** или индексе онлайн-справки, ищущей ключевое слово, определенное в поле **Help ID**.
Раздел **Info**		–
	Список **Language**	Содержит языки, которые доступны для графического интерфейса пользователя SoMachine. Если Вы выбираете язык, содержание зависимых от языка элементов **Comment (Комментария)**, **Description (Описания)** и **Picture** (**Рисунка)** выводится на экран на выбранном языке. Если никакое специфичное для языка содержание не доступно, по умолчанию выведен **English (**английский язык).
	Кнопка **Import From File**	Выводит на экран стандартное диалоговое окно **Open**. Оно позволяет Вам искать XML-файл, который содержит локализованное содержание зависимых от языка элементов **Comment**, **Description и Picture**. Структура этого XML-файла должна следовать структуре, обозначенной в примере *(см. стр. 741)*.
	Поле **Comment**	Позволяет Вам вводить короткий текст (например, чтобы обеспечить обзор содержания и цель выбранной библиотеки или библиотеки шаблонов). Этот текст обозначен как подсказка, когда Вы выбираете библиотеки шаблонов в SoMachine.
	Поле **Description**	Позволяет Вам вводить длинный текст (например, чтобы представить подробное описание содержания и цель выбранной библиотеки или библиотеки шаблонов).
	Кнопка **Picture parameter ...**	Позволяет Вам находить путь к специфичной для языка картине. Вы можете также щелкнуть кнопку **...** для просмотра графического файла. Поддерживаемые графические форматы: ● Bitmap: **.bmp* ● JPEG: **.jpg* ● Graphics interchange format: **.gif* ● Icon: **.ico* После того, как картина была определена, она будет выведена на экран в диалоговом окне **Properties**. Если Вы щелкаете по кнопке **OK**, картина встраивается в шаблон

Флажок **Read-Only** является доступным только для библиотек шаблонов, чтобы указывать, является ли выбранная библиотека шаблонов в состоянии только для чтения. Здесь невозможно изменить состояние библиотеки шаблонов.

Локализация языковозависимых элементов

Вы можете локализовать содержание языковозависимых элементов **Comment**, **Description** и **Picture,** импортировав XML-файл со следующей структурой:

```
<?xml version="1.0" encoding="UTF-8"?>
<TemplateProperties>
<HelpId>SoMProg.chm::/SoMProg_D-SE-0001286.htm#D-SE-0001286.1</HelpId>
<PropertySet languageId = "en">
<Comment>This is a short description</Comment>
<Description>This is a long description</Description>
<ImageFile>PictureEnglish.jpg</ImageFile>
</PropertySet>
<PropertySet languageId = "de">
<Comment>Kurze Beschreibung</Comment>
<Description>Lange Beschreibung</Description>
<ImageFile>PictureGerman.jpg</ImageFile>
</PropertySet>
</TemplateProperties>
```

Глава 33

Управление шаблонами устройства

Раздел 33.1

Управление шаблонами устройств

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Факты шаблонов устройств	745
Добавление устройств из шаблона	746
Создание шаблона устройства на основе полевых устройств или модулей Вх/Вых	749
Визуализации, подходящие для создания шаблонов устройств	750
Дополнительная информация об интегрировании управляющей логики в шаблоны устройств	751
Шаги для создания шаблона устройства	753

Факты шаблонов устройств

Общая информация об использовании терминов

Следующее описание применяется к полевым устройствам, а также к модулям Вх/Вых даже при том, что только термин полевого устройства используется, чтобы увеличить удобочитаемость.

Содержание шаблонов устройств

Шаблоны устройства связаны с определенным полевым устройством или модулем Вх/Вых. Они содержат следующую информацию:

- Конфигурация полевой шины
- Управляющая логика (программирование контроллера) (дополнительно)
- Элементы визуализации (программирование визуализации) (дополнительно)

Использование шаблонов устройств

Уже доступные шаблоны устройств сохранены в библиотеках шаблонов. Каждая библиотека шаблонов содержит определение для нескольких шаблонов устройств, у которых есть общая база (связана с блоком управления приводом, например).

Вы можете их выбрать и адаптировать к требованиям Ваших индивидуальных проектов SoMachine, чтобы создать новые предварительно сконфигурированные и готовые к использованию полевые устройства.

Создание новых шаблонов устройства

Чтобы сделать Ваши уже сконфигурированные полевые устройства повторно используемыми для любого проекта SoMachine, сохраните их как шаблоны устройства. Это также включает программирование контроллера и визуализацию, связанные с этим полевым устройством.

Версии шаблонов устройства

Во время создания шаблона устройства производится проверка, существует ли описание устройства для создаваемого устройства, на самом деле. Если этого нет, устройство автоматически обновляется до последней версии, если более поздняя версия уже существует.

Добавление устройств из шаблона

Обзор

Шаблоны устройства связаны с определенным устройством полевой шины. Они содержат следующую информацию:

- Конфигурация устройства полевой шины
- Управляющая логика (программирование контроллера) (дополнительно)
- Элементы визуализации (программирование визуализации) (дополнительно)

Вы можете создать свои собственные шаблоны устройства из Вашего проекта. Для получения дополнительной информации обратитесь к главе *Шаги создания шаблона устройства* *(см. стр. 753)*.

Добавление устройства из шаблона

SoMachine обеспечивает 2 способа добавить устройство из шаблона устройства:

- Создание устройства с использованием шаблона устройства перетаскиванием:

Шаг	Действие
1	Откройте вид **Field Devices** аппаратного каталога.
2	Внизу вида **Field Devices** активируйте опцию **Device Template**. **Результат**: Шаблоны полевых устройств, доступных в SoMachine, отображаются в виде **Field Devices**.
3	Выберите запись в виде **Field Devices**, перетащите её к дереву **Devices tree** и освободите её в подходящем подузле контроллера. **Комментарий**: Подходящие подузлы выделены SoMachine. **Результат**: Отображается диалоговое окно **Add Device From Template**.
4	В диалоговом окне **Add Device From Template** устанавите имя устройства (**Device Name)**, а также адрес устройства (**Device Address)**, если полевая шина требует числовых адресов. В случае, если шаблоны устройства включают управляющую логику, выберите программу (POU), в которую вставлена управляющая логика.
5	Щелкните по кнопке **OK**. **Результат**: устройство создано и параметризовано согласно выбранному шаблону устройства, включая дополнительные экраны визуализации и управляющую логику.

- Создание устройства с использованием шаблона устройства через контекстное меню:

Шаг	Действие
1	Откройте дерево **Devices tree**.
2	Щелкните правой кнопкой по менеджеру полевого устройства и выполните команду **Add Device From Template** из контекстного меню. **Результат**: Отображается диалоговое окно **Add Device From Template**.
3	В диалоговом окне **Add Device From Template** выберите **Device Template**, которое будет использоваться и установите имя устройства (**Device Name)**, а также адрес устройства (**Device Address)**, если полевая шина требует числовых адресов. В случае, если шаблоны устройства включают управляющую логику, выберите программу (POU), в которую вставлена управляющая логика.
4	Щелкните по кнопке **OK**. **Результат**: устройство создано и параметризовано согласно выбранному шаблону устройства, включая дополнительные экраны визуализации и управляющую логику.

ПРИМЕЧАНИЕ: функция отмены / восстановления не доступны для процесса создания полевых устройств.

Именование объектов

Во избежание конфликтов имен, если один и тот же шаблон устройства используется в качестве основания для создания различных полевых устройств, следующие соглашения о присвоении имен применяются к полевым устройствам и связанным объектам (FB, визуализация и переменные):

Если имя исходного объекта...	Тогда...
Случай 1:	
содержит имя исходного полевого устройства,	эта часть объекта заменяется именем нового полевого устройства, которое создается.
Пример:	
Шаблон устройства для полевого устройства `ATV1` содержит переменную `Var_ATV1_Input`.	Для нового устройства `Axis1`, создаваемого с этим шаблоном устройства, новую переменную соответственно называют `Var_Axis1_Input`.
Случай 2:	
не содержит имя исходного устройства,	имя нового устройства плюс подчеркивание вставляется в исходном имени, чтобы сформировать уникальное новое имя.
Пример:	
Шаблон устройства для полевого устройства `ATV1` содержит переменную `Var_Input1`.	Для нового устройства `Axis1`, создаваемого с этим шаблоном устройства, новую переменную соответственно называют `Axis1_Var_Input1`

Создание шаблона устройства на основе полевых устройств или модулей Вх/Вых

Обзор

Вы можете создать шаблоны устройств на основе полевых устройств или модулей Вх/Вых. Следующее описание применяется к полевым устройствам, а также к модулям Вх/Вых даже при том, что термин полевого устройства используется только для того, чтобы увеличить удобочитаемость.

Последующие параграфы перечисляют:

- Критерии, которые должны быть выполнены, чтобы сохранить полевое устройство или модуль Вх/Вых, включая логику и визуализацию, как шаблон устройства;
- Информацию, которая сохраняется в шаблоне устройства.

Предпосылки для полевых устройств

Полевые устройства должны соответствовать следующим критериям, чтобы быть сохранёнными как шаблоны устройств:

- Полевые устройства должны быть связаны с полевыми шинами, перечисленными в списке поддерживаемых полевых шин *(см. стр. 735)*;
- Тип устройства должен быть установлен в **Device Repository**.

Предпосылки для модулей Вх/Вых

Только поддерживаемые модули Вх/Вых могут быть сохранены как шаблоны *(см. стр. 735)*.

Предпосылки для приложения

Вы можете создать шаблоны только из корректных приложений. Корректный означает, что ошибки не обнаружены во время процесса **Build** (сборки).

Предпосылки для включения управляющей логики в шаблон

Чтобы включить управляющую логику в шаблон, требуется, чтобы управляющая логика содержала один или несколько разделов кода, которые обмениваются данными с этим полевым устройством. Эта управляющая логика должна быть выполнена (добавлена к задаче, или вызвана другой программой). Иначе ее не рассматривают при выполнении команды **Build**.

Информация устройства, сохраненная в шаблонах устройства

Следующая информация полевых устройств сохраняется в шаблонах устройств:

- Конфигурация устройства
- Распределение Вх/Вых полевого устройства
- Визуализации, которые подходят для полевого устройства
- Данные обмена управляющей логики с полевым устройством

Визуализации, подходящие для создания шаблонов устройств

Обзор

Каждый шаблон устройства может быть связан с 1 или более визуализациями Logic Builder. Поддерживаемые типы визуализации описываются следующим образом.

Поддерживаемые визуализации

SoMachine поддерживает оба типа визуализаций:

- Простые (Plain) визуализации
- Модульные визуализации с использованием фреймов

Визуализации с использованием фреймов имеют большую гибкость и модульность.

Простые визуализации

Визуализации без фреймов основываются на единственном объекте визуализации, создаваемом для устройства Вх/Вых.

SoMachine ссылается на данные устройства Вх/Вых в свойствах визуальных элементов. Когда Вы создаете новое устройство на основе этого шаблона устройства, SoMachine непосредственно заменяет переменные в свойствах визуальных элементов.

Визуализации с использованием фреймов

Визуализация с использованием фреймов построена из основного экрана, который может быть заполнен другими визуализациями, используя группу меньших визуализаций, комбинируемых как модули в предопределенных областях основного экрана (фреймах).

В основном экране объект фрейма помещен как прямоугольный объект как контейнер. Вы можете присвоить другую визуализацию такому контейнеру.

Встроенная визуализация тогда может использоваться с интерфейсом, чтобы получить внутренне доступ к визуальным элементам.

Для получения дополнительной информации обратитесь к части **Programming with SoMachine** → **Visualization** онлайн-справки SoMachine.

Для использования встроенных визуализаций для шаблонов устройств определите интерфейс, который содержит определение всех переменных, связанных с соединением к устройству Вх/Вых или функциональным блоком для каждого модуля визуализации. Когда Вы создаёте новое устройство, основанное на этом шаблоне устройства SoMachine адаптирует все метки-заполнители встроенных в визуализации согласно созданному имени устройства Вх/Вых.

ПРИМЕЧАНИЕ: Вся визуализации с использованием фреймов и функциональных блоков, связанные с определенным устройством Вх/Вых, должны быть определены в библиотеке так, чтобы SoMachine могло найти их.

Дополнительная информация об интегрировании управляющей логики в шаблоны устройств

Обзор

Вы можете включать управляющую логику в шаблон устройства, если логика содержит один или несколько разделов кода, которые обмениваются данными с этим полевым устройством одним из следующих способов:

- Раздел кода использует новую переменную, которая определена в распределении Вх/Вых полевого устройства.
- Раздел кода и распределение Вх/Вых полевого устройства используют общую переменную, которая определена в GVL или программе контроллера, содержавшей приложение, которому принадлежит раздел кода.

  ПРИМЕЧАНИЕ: Если Вы используете структуры или массивы, проверяйте, что они связаны только с единственным полевым устройством.
- Раздел кода и полевое устройство используют фиксированную специфическую для устройства переменную (например, переменные задания по осям, используемые с приводами Altivar или Lexium).

Внутренне соединенные вызовы разделов кода

Разделы кода состоят из последовательности внутренне соединенных вызовов функциональных блоков, функций и операторов.

Если одно из следующих соотношений существует между отдельными вызовами, их рассматривают как соединяемые:

- графическое соединение существует между отдельными вызовами в CFC, FBD и LD
- переменная соединена с выходом одного вызова и входом другого вызова
- Один вызов использует параметр другого вызова

Индивидуально выбираемые функциональные блоки

Вы можете индивидуально выбрать функциональные блоки, включенные в те разделы кода, которые обмениваются данными с полевым устройством, и которое будет включено в шаблон устройства. Это позволяет Вам создавать различные шаблоны устройства, обеспечивающие различные функции для одного и того же полевого устройства.

ПРИМЕЧАНИЕ: Тип функционального блока должен быть определен в библиотеке.

Вставка выражений в шаблоны устройства

Выражения, а также переменные, использованные в этих выражениях, которые соединены с параметрами функционального блока, функции или оператора автоматически сохраняются в шаблоне устройства.

Общие методы для создания управляющей логики

Включайте в шаблон устройства только простую управляющую логику.

Таким образом, разделы кода работают тождественно, даже если они создаются на различных языках IEC.

ПРИМЕЧАНИЕ: Для сложной управляющей логики Вы должны, скорее всего, создать шаблон функции.

Методы создания управляющей логики в FBD / LD

Избегайте элементов обнаружения фронтов, потому что они не существуют на других языках IEC.

Если возможно, используйте вместо этого функциональные блоки R_TRIG или F_TRIG.

Методы создания управляющей логики в CFC

Используйте команду **Execution Order** → **Order By Data Flow**, чтобы упорядочить элементы CFC, принадлежащие тому же разделу кода, согласно их позиции в потоке данных. Это обеспечивает лучшую совместимость с другими языками IEC.

Обеспечивайте пространство (в горизонтальном направлении) между отдельными элементами CFC, потому что из-за переименования при создании нового устройства из шаблона, имена переменных расширяются.

Пример управляющей логики

Следующая иллюстрация показывает типичный пример раздела кода для устройства распределённых Вх/Вых Advantys OTB в приложении:

Раздел кода состоит из следующих функциональных блоков:

Имя	Тип	Функция
`InOTB_Conv20`	Входной блок	Преобразование данных, прибывающих из OTB в формат, требуемый блоком управления
`Conv20`	Блок управления	Обработка данных
`OutOTB_Conv20`	Выходной блок	Преобразование данных, прибывающих из блока управления, в формат, требуемый OTB

Переменные InByte1_Conv20, InByte2_Conv20 и OutByte_Conv20 определены в распределении устройств ввода-вывода OTB. Это означает, что раздел кода обменивается данными с устройством OTB. Оно может таким образом стать частью шаблона устройства.

Шаги создания шаблона устройства

Обзор

В последующих параграфах перечислены шаги, которые должны быть выполнены, чтобы сохранить полевые устройства, соответствующие критериям, утвержденным в *Создание шаблона устройства на основе полевых устройств или модулей Вх/Вых (см. стр. 749)*.

Шаги для сохранения полевого устройства как шаблона

Чтобы сохранить уже существующее полевое устройство как шаблон устройства, выполните следующее:

Шаг	Действие
1	Щелкните правой кнопкой по полевому устройству, которое Вы хотите сохранить как шаблон устройства в дереве **Devices tree**.
2	Выберите команду **Save As Device Template** из контекстного меню. **Результат**: SoMachine автоматически создает приложение. После того, как процесс построения был успешно завершен, диалоговое окно **Save as Device Template** будет выведено на экран.
3	Определите новый шаблон устройства в диалоговом окне **Save as Device Template**, как указано ниже.
4	Нажмите "**OK**", чтобы закрыть диалоговое окно **Save as Device Template** и создать Ваш новый шаблон устройства.

Диалоговое окно сохранения устройства как шаблона

Диалоговое окно **Save As Device Template** содержит следующие параметры:

1 указывает тип полевого устройства, на котором базируется шаблон устройства
2 указывает тип полевой шины полевого устройства
3 имя шаблона устройства, который будет создаваться (первоначально имя исходного полевого устройства)
4 выбор библиотеки шаблонов, к которой будет добавлен шаблон устройства
5 выбор функциональных блоков и визуализаций, которые должна быть сохранены с шаблоном устройства
6 кнопка **Properties**, чтобы ввести дополнительную информацию к шаблону устройства

Определение имени нового шаблона устройства

Используйте текстовое поле **Template Name**, чтобы определить имя для Вашего шаблона устройства.

По умолчанию это текстовое поле содержит имя выбранного полевого устройства.

Вы можете или ввести имя по Вашему выбору непосредственно в это текстовое поле, или Вы можете щелкнуть кнопку**...**, чтобы выбрать существующий шаблон устройства из списков, если Вы хотите перезаписать шаблон этого устройства.

Выбор библиотеки шаблонов

Чтобы выбрать одну из ранее установленных или созданных библиотек шаблонов, в которых должен быть сохранен шаблон устройства, выполните следующее:

Шаг	Действие
1	В диалоговом окне **Save as Device Template** щелкните кнопку **...** справа от текстового поля **Template Library**. **Результат**: Будет выведено диалоговое окно **Select Template Library**.
2	Диалоговое окно **Select Template Library** отображает все библиотеки шаблонов, которые были установлены для текущего проекта или были созданы. Библиотеки шаблонов, защищенные от записи, не отображаются. Чтобы добавить новый шаблон устройства к одной из этих библиотек шаблонов, выберите подходящую запись и нажмите **OK**.

Выбор функциональных блоков

Для выбора экземпляров функционального блока, которые будут включены в шаблон устройства, выполните следующее:

Шаг	Действие
1	В диалоговом окне **Save as Device Template** щелкните кнопку **...** справа от текстового поля **Function Blocks**. **Результат**: Будет выведено диалоговое окно **Select Function Block**. Диалоговое окно **Select Function Block** отображает все экземпляры функционального блока, содержащиеся в управляющей логике полевого устройства *(см. стр. 751)*.
2	Установите флажок индивидуального функционального блока, чтобы выбрать его для шаблона устройства. Или установите флажок корневого узла, чтобы выбрать все элементы ниже этого узла.
3	Щелкните по кнопке **OK**.

Выбор визуализации

Для выбора визуализаций, которые включены в полевое устройство, выполните следующее:

Шаг	Действие
1	В диалоговом окне **Save as Device Template** щелкните кнопку **...** справа от текстового поля **Visualizations**. **Результат**: Будет выведено диалоговое окно **Select Visualizations**. Диалоговое окно **Select Visualizations** отображает те визуализации, которые связаны с полевым устройством или с одним из выбранных функциональных блоков.
2	Установите флажок индивидуальной визуализации, чтобы выбрать её для шаблона устройства. Или установите флажок корневого узла, чтобы выбрать все элементы ниже этого узла.
3	Щелкните по кнопке **OK**.

Добавление дополнительной информации к новому шаблону устройства

Для добавления дополнительной информации к новому шаблону устройства, нажмите кнопку **Properties...**. Открывается диалоговое окно **Properties**. Оно позволяет Вам вводить дополнительную информацию для шаблона устройства. Так как диалоговое окно идентично для шаблонов устройства и библиотек шаблонов, смотрите описание в разделе *Добавление дальнейшей информации для шаблонов или библиотек (см. стр. 739)*.

Глава 34

Управление шаблонами функций

Раздел 34.1

Управление шаблонами функций

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Факты шаблонов функций	759
Добавление функций из шаблона	760
Прикладные функции как основа для шаблонов функций	766
Шаги для создания шаблона функции	768

Факты шаблонов функций

Содержание шаблонов функций

Шаблоны функций представляют выделенную функциональность управления и визуализации, которые связаны с прикладной функцией.

Шаблон функции может включать следующие элементы:

- Одна или несколько программ IEC
- Одно или несколько полевых устройств или модулей Вх/Вых, которые используются прикладной функцией
- Одна или несколько визуализаций, которые используются, чтобы визуализировать прикладную функцию
- Один или несколько списков глобальных переменных
- Одна или несколько глобальных переменных, которые могут быть совместно использованы с другими прикладными функциями
- Одна или несколько трассировок
- Одна или несколько таблиц CAM
- Одна или несколько переменных Вх/Вых, которые отображены на канале Вх/Вых
- Один или несколько параметров шаблона

Использование шаблонов функций

Уже доступные шаблоны функций сохранены в библиотеках шаблонов. Каждая библиотека шаблонов содержит определение для нескольких шаблонов функций, у которых есть общая база (например, все связаны с упаковочными приложениями).

Вы можете легко выбрать их и адаптировать к требованиям Ваших индивидуальных проектов SoMachine, чтобы создать новые готовые к использованию прикладные функции.

Создание новых шаблонов функций

Чтобы сделать Вашу уже созданную прикладную функцию повторное используемой для любого проекта SoMachine, Вы можете сохранить её как шаблон функции.

Когда Вы сохраняете шаблон функции, решите, в какой библиотеке шаблонов он должен быть сохранен.

Версии шаблонов функций

Во время создания шаблона функции выполняется проверка, существует ли на самом деле описание устройства для устройства, которое создается. Если его нет, устройство автоматически обновляется до последней версии, если существует более поздняя версия.

Добавление функций из шаблона

Процедура

SoMachine обеспечивает 2 способа добавления функции из шаблона функций:

Чтобы добавить прикладную функцию из шаблона функций через перетаскивание, выполните следующее:

Шаг	Действие
1	Откройте вид **Macros** из **Software Catalog**.
2	Выберите шаблон функции из вида **Macros**, перетащите его к **Applications tree** и освободите его в подходящем узле **Application** или папке ниже узла **Application**. **Примечание**: Подходящие узлы выделены SoMachine. **Результат**: Открывается диалоговое окно **Add Function From Template**.

Чтобы добавить прикладную функцию из шаблона функции через контекстное меню, выполните следующее:

Шаг	Действие
1	Откройте **Applications tree.**
2	Щелкните правой кнопкой по узлу **Application** или папке ниже узла **Application**, и выполните команду **Add Function From Template** из контекстного меню. **Результат**: отображается диалоговое окно **Add Function From Template**.

Диалоговое окно добавления функции из шаблона

Диалоговое окно **Add Function From Template** обеспечивает следующие элементы, чтобы сконфигурировать Вашу функцию:

Элемент		Описание
Текстовое поле **Function Name**		Введите имя, которое используется для новой папки этого приложения, и для наименования элементов, которые она содержит.
Шаблон функции		Щелкните кнопку **...** и выберите шаблон функции из диалогового окна **Select Function Template**.
Таблица **I/O Devices (устройств Вх/Вых)**		–
	Device Name	Содержит имя будущего полевого устройства. Вы не можете изменить это имя.
	Device Type	Указывает тип полевого устройства. Вы не можете отредактировать эту ячейку.
	Fieldbus Type	Указывает тип полевой шины полевого устройства. Вы не можете отредактировать эту ячейку.

Элемент		Описание
	Master	Содержит ведущее устройство полевой шины (master), с которым соединено полевое устройство. Если есть несколько ведущих устройств, Вы можете выбрать ведущее устройство по Вашему выбору из списка.
	Address	Первоначально пустой. Для полевых устройств на полевых шинах, которые требуют числовых адресов (Modbus serial line и CANopen), щелкните кнопку ... справо от поля и присвойте адрес по Вашему выбору.
Таблица **I/O Mapping**		Списки переменных Вх/Вых, которые являются частью шаблона функции. Это позволяет Вам отображать их на каналах Вх/Вых существующих устройств и модулей.
	Name	Содержит имя переменной Вх/Вых, которая должна быть отображена на канале ввода-вывода.
	Data Type	Указывает тип данных канала Вх/Вых, на котором была первоначально отображена переменная Вх/Вых.
	Mapping	Щелкните кнопку ..., чтобы открыть диалоговое окно **Select I/O Mapping**. Оно позволяет Вам выбрать канал Вх/Вых, на котором Вы можете отобразить выбранную переменную. После того, как переменная была отображена на канале Вх/Вых, это поле **Mapping** содержит адрес входа или выхода канала Вх/Вых, на котором отображена переменная.
	Description	Содержит описание переменной Вх/Вых.
Таблица **Parameters**		Перечисляет параметры шаблона, включенные в шаблон функции.
	Object	Указывает имя GVL или программы, в которой определена переменная. Вы не можете отредактировать это поле.
	Name	Содержит имя переменной. Вы не можете отредактировать эту ячейку.
	Data Type	Указывает тип данных переменной. Вы не можете отредактировать эту ячейку.
	Default	Указывает значение переменной по умолчанию. Это - начальное значение переменной, когда шаблон создавался. Вы не можете отредактировать эту ячейку.
	New Value	Отредактируйте эту ячейку, если Вы хотите присвоить новое значение переменной. Если Вы оставляете эту клетку пустой, для этой переменной используется значение по умолчанию. Введите значение, которое допустимо для определенного типа данных.
	Описание	Содержит описание переменной.
Кнопка **OK**		Подтвердите свои параметры настройки, щелкнув по кнопке **OK**. **Результат**: SoMachine проверяет, корректны ли параметры настройки, и вставляет новую прикладную функцию как отдельный узел ниже узла **Application** или выводит на экран сообщение об обнаружении ошибок.

Диалоговое окно выбора размещения Вх/Вых

Диалоговое окно **Select I/O Mapping** используется, чтобы отобразить переменную, выбранную в диалоговом окне **Add Function From Template** на канал Вх/Вых.

Оно отображает доступные каналы Вх/Вых в древовидной структуре, подобной **Devices tree**. Корневой узел - контроллер. Отображаются только те каналы Вх/Вых, чей тип данных соответствует типу данных новой переменной.

2 типа данных совместимы, если у них есть идентичные имена типов или если они - элементарные типы данных IEC того же размера.

Пример:

UINT --> INT разрешено

UDINT --> INT не разрешено

Отображение подузлов щелчками по знакам "плюс".

Диалоговое окно **Select I/O Mapping** содержит следующие колонки:

Колонка	Описание
Channel	Содержит древовидную структуру. Каждое устройство представлено именем устройства и значком устройства. Каждый канал Вх/Вых представлен именем канала.
Address	Содержит адрес входа / выхода, который соответствует каналу Вх/Вых.
Mapping	Содержит переменную Вх/Вых, которая в настоящее время отображена на канале Вх/Вых.
Description	Содержит описание канала Вх/Вых.

Рассмотрите следующие правила, применяющиеся для отображения переменных на каналах Вх/Вых:

- Размещайте все переменные, обеспеченные шаблоном функции, на каналы Вх/Вых.
- Вы можете разместить переменную Вх/Вых шаблона функции на канал Вх/Вых, у которого уже есть отображение. Существующее отображение будет перезаписано.
- Любые размещения, которые приводят к многократным присвоениям переменных на одном и том же канале Вх/Вых, не позволены.

Создаваемые объекты

Шаблон функции создает следующие объекты в Вашем проекте:

Объект	Описание
Root folder (Корневая папка)	Новая папка создается под узлом **Application** в виде **Devices**, которую называют, как определено в текстовом поле **Function Name**.
Field devices (Полевые устройства)	Полевые устройства, которые включены в шаблон функции, создаются с использованием имен, к которым применяются правила наименования, и соединены с ведущим устройством полевой шиной. Распределение Вх/Вых при необходимости автоматически корректируется.
Visualizations (Визуализации)	Визуализации, которые включены в шаблон функции, создаются ниже корневой папки, с использованием имен, к которым применяются правила наименования. Свойства визуализации автоматически корректируются.
Programs Программы	Программы, которые включены в шаблон функции, создаются ниже корневой папки, с использованием имен, к которым применяются правила наименования. Имена тех объектов в программе, которые являются частью шаблона функции, корректируются автоматически.
Traces (Трассировки)	Трассировки, которые включены в шаблон функции, создаются ниже корневой папки, с использованием имен, к которым применяются правила наименования, и могут использоваться, чтобы проследить переменные, принадлежащие прикладной функции.
CAM tables (Таблицы CAM)	Таблицы CAM, которые включены в шаблон функции, создаются ниже корневой папки, с использованием имен, к которым применяются правила наименования. Они требуются, если только прикладная функция включает устройства SoftMotion.
Task configuration (Конфигурация задач)	Конфигурация задачи корректируется как требуется шаблоном функции.
Global variable lists (Списки глобальных переменных)	Списки глобальных переменных, которые включены в шаблон функции, создаются ниже корневой папки, с использованием имен, к которым применяются правила именования.

Объект	Описание
External variables (Внешние переменные)	Глобальные переменные, списки глобальных переменных которых не принадлежат шаблону функции, восстанавливаются в их исходном списке глобальных переменных следующим образом: Если список глобальных переменных с оригинальным именем уже не существует ниже приложения, он создается автоматически. Если глобальная переменная с оригинальным именем уже не существует в этом списке глобальных переменных, она создается автоматически. Если тип глобальной переменной не корректен, SoMachine выдает сообщение об обнаружении ошибок.
Persistent variables (Персистентные переменные)	Персистентные переменные восстанавливаются в соответствующем списке переменных приложения следующим образом: Если список персистентных переменных уже не существует ниже приложения, он создается автоматически с его оригинальным именем. Если переменная с оригинальным именем уже не существует в списке персистентных переменных, она создается автоматически. Если тип персистентной переменной не корректен, SoMachine выдает сообщение.

Любые объекты, которые созданы при реализации шаблона функции, перечислены в панели **Messages.**

Наименование объектов

Во избежание конфликтов имен, если Вы несколько раз реализуете один и тот же шаблон функции на одном и том же устройстве контроллера, следующие соглашения о присвоении имен применяются к прикладным функциям и связанным объектам:

Если имя исходного объекта...	Тогда ...
Случай 1:	
содержит имя прикладной функции,	эта часть объекта заменяется названием новой прикладной функции, которая создается.
Пример:	
Шаблон исходной прикладной функции `Axis` содержит программу `Axis_Init`.	Для новой прикладной функции `Axis1`, создаваемой с этим шаблоном, новую программу соответственно называют `Axis1_Init`.
Случай 2:	
не содержит имя прикладной функции,	имя новой прикладной функции плюс подчеркивание вставляется в оригинальном имени, чтобы сформировать уникальное новое имя.
Пример:	
Исходная прикладная функция `Axis` содержит программу `InitProg`.	Для новой прикладной функции `Axis1`, создаваемой с этим шаблоном функции, новую программу соответственно называют `Axis1_InitProg`.

ПРИМЕЧАНИЕ: Используйте довольно краткие имена для Ваших прикладных функций так, чтобы они отображались полностью.

Прикладные функции как основа для шаблонов функций

Обзор

Последующие параграфы перечисляют:

- Критерии, которые должны быть выполнены, чтобы сохранить прикладную функцию с ее связанными полевыми устройствами, модулями Вх/Вых и визуализациями как шаблон функции;
- Информацию, которая сохраняется в шаблоне функции.

Определение прикладных функций как шаблонов функций

Вы можете сохранить прикладные функции как шаблоны функций, щелкнув правой кнопкой по подузлу Вашего узла приложения (**Application**) в **Applications tree**. Или Вы можете создать свой собственный шаблон в виде **Macros**, выбрав отдельные объекты для Вашего шаблона. Эти 2 процедуры описаны в разделе *Шаги по созданию шаблона функции (см. стр. 768)*.

Предпосылки для приложения

Вы можете создать шаблоны только из корректных приложений. Корректный означает, что ошибки не обнаружены во время процесса сборки.

Предпосылки для сохранения прикладной функции как шаблона функции

Чтобы сохранить прикладную функцию как шаблон функции, предпосылка - то, что все программы прикладной функции выполняются.

Это означает, что они должны соответствовать одному из следующих критериев:

- Они должны быть добавлены к задаче.
- Их должна вызвать другая программа.

Иначе их не будут рассматриваться при выполнении команды **Build**.

Переменные Вх/Вых в шаблонах функций

Переменная Вх/Вых - переменная, которая отображена на канале Вх/Вых полевого устройства. Она сохранена в шаблоне функции, если применены следующие условия:

- Переменная Вх/Вых используется любой программой или визуализацией, которая включена в шаблон функции.
- Полевое устройство или модуль Вх/Вых, на котором отображена переменная Вх/Вых, не могут быть включены в шаблон функции.

Вы можете отобразить переменную Вх/Вых, которая сохранена в шаблоне функции на существующем канале Вх/Вых, когда прикладная функция создается из шаблона функции *(см. стр. 761)*.

Переменная Вх/Вых имеет описание, которое отображается в диалоговом окне **Add Function From Template**.

Это описание создается следующим образом:
- Если переменная ввода-вывода была создана заново на вкладке **I/O Mapping** редактора устройства *(см. стр. 138)*, описание взято из описания канала ввода-вывода (это применяется только тогда, если исходное описание было изменено).
- Если переменная ввода-вывода является ссылкой на существующую переменную, описание взято из комментария этой переменной.

Параметры шаблона

Параметр шаблона - переменная с корректируемым начальным значением.

Пример: Когда устройство используется через коммуникационный функциональный блок, тогда Вы должны присвоить адрес устройства этому функциональному блоку как входной параметр. Чтобы быть в состоянии установить этот адрес, соедините переменную с функциональным блоком и определите переменную как параметр шаблона.

Переменная может стать параметром шаблона, если применены следующие условия:
- Переменная определена в программе или списке глобальных переменных, которые включены в шаблон функции.
- Переменная имеет простой тип данных (BOOL, любой тип числовых данных, любая STRING, типы псевдонима на основе простого типа данных).
- Начальное значение переменной определено явно как литеральное значение.

Все переменные, которые удовлетворяют этим условиям, могут быть выбраны как параметр шаблона, когда шаблон функции сохраняется *(см. стр. 771)*.

Если переменная была выбрана как параметр шаблона, начальное значение этой переменной может быть скорректировано, когда создается новая прикладная функция из шаблона функции *(см. стр. 761)*.

Объекты, сохраняемые в шаблонах функций

Следующие объекты сохраняются в шаблонах функций:
- Все программы, которые расположены непосредственно в папке прикладной функции, а также их подобъекты
- Все списки глобальных переменных, которые расположены непосредственно в папке прикладной функции
- Все визуализации, которые расположены непосредственно в папке прикладной функции
- Все таблицы CAM, которые расположены непосредственно в папке прикладной функции
- Все трассировки, которые расположены непосредственно в папке прикладной функции
- Все полевые устройства и модули Вх/Вых, которые используются любой программой или визуализацией, которые включены в шаблон функции
- Все глобальные переменные, списки переменных которых не являются частью шаблона функции, но которые используются любой программой или визуализацией, которая является частью шаблона функции
- Все персистентные переменные, которые используются любой программой или визуализацией, которая является частью шаблона функции

ПРИМЕЧАНИЕ: Любые другие типы объектов не сохраняются в шаблоне функции (даже если они сохранены в папке прикладной функции). Используйте только функциональные блоки и типы данных, которые сохранены в библиотеке.

Шаги по созданию шаблона функции

Обзор

SoMachine обеспечивает 2 способа создания шаблона функции:

- Из вида **Macros**, используя диалоговое окно **Create New Template**.
- Из **Applications tree,** используя диалоговое окно **Save as Function Template**.

Следующие параграфы перечисляют шаги, которые должны быть выполнены, чтобы сохранить доступные прикладные функции, которые соответствуют критериям, утвержденным в *Прикладные функции как основа шаблонов функций (см. стр. 766)*.

Процедура через вид Макросов

Процедура через вид **Macros** позволяет Вам создавать свой собственный шаблон функции перетаскиванием элементов:

Шаг	Действие
1	В виде **Macros** разверните раздел **My Template**.
2	Выберите узел **My Template** и щелкните по кнопке зеленого плюса. **Результат**: новый узел с именем по умолчанию **LIB1** вставлен ниже узла **MyTemplate**.
3	Выберите узел **LIB1** и щелкните по кнопке зеленого плюса. **Результат**: Отображается диалоговое окно **Create New Template**.

Шаг	Действие
4	На вкладке **Function Template** диалогового окна **Create New Template** введите имя (**Name)** для своего шаблона функции. Перетащите элементы, которые Вы хотите включить в шаблон функции, от **Applications tree** в поле **Elements** вкладки **Function Template**. Элементы, перечисленные в этом поле, вставляются в Ваш шаблон функции. **ПРИМЕЧАНИЕ**: элементы должны принадлежать одному и тому же приложению.
5	Вкладка **Parameters** диалогового окна **Create New Template** отображает те переменные, которые включены в элементы, выбранные Вами на вкладке **Function Template**. Из списка переменных выберите те, которые Вы хотите объявить как параметры шаблона, устанавливая флажок переменной или узла.
6	Вкладка **Properties** диалогового окна **Create New Template** позволяет Вам добавлять дополнительную информацию к шаблону функции. Вы можете вставить ссылку к онлайн-справке этого шаблона функции. Диалоговое окно позволяет Вам добавлять далее текстовую информацию, которая может быть локализована, и Вы можете добавить графическое иллюстрирование этого шаблона функции. Для описания этих параметров обратитесь к разделу *Добавление дополнительной информации в шаблоны или библиотеки шаблонов* *(см. стр. 739)*.
7	Нажмите кнопку **Save Template**.

Процедура через Applications tree

Чтобы сохранить уже доступную прикладную функцию как шаблон функции, выполните следующее:

Шаг	Действие
1	Щелкните правой кнопкой по подпапке Вашего узла **Application** в **Applications tree**.
2	Выберите команду **Save As Function Template** из контекстного меню. **Результат**: SoMachine автоматически строит приложение. После того, как процесс построения был успешно завершен, диалоговое окно **Save As Function Template** будет выведено на экран.
3	Определите новый шаблон функции, как утверждено ниже.
4	Нажмите **OK**, чтобы закрыть диалоговое окно **Save as Function Template** и создать Ваш новый шаблон функции. **Результат**: SoMachine проверяет, что шаблон функции может быть создан и выводит на экран сообщение, что шаблон функции был создан успешно или указывает обнаруженные ошибки.

Присвоение имени шаблона

В текстовом поле **Template Name** диалогового окна **Save as Function Template** определите имя, под которым шаблон функции сохраняется в библиотеке шаблонов. По умолчанию это текстовое поле содержит имя папки, содержащей Вашу прикладную функцию в **Applications tree**, но Вы можете адаптировать имя к своим индивидуальным требованиям.

Выбор библиотеки шаблонов

Чтобы выбрать одну из ранее установленных или создаваемых библиотек шаблонов, в которых должен быть сохранен Ваш новый шаблон функции, выполните следующее:

Шаг	Действие
1	В диалоговом окне **Save as Function Template** щелкните кнопку **...** рядом с текстовым полем **Template Library**. **Результат**: Отображается диалоговое окно **Select Template Library**.
2	Диалоговое окно **Select Template Library** отображает все библиотеки шаблонов, которые были установлены для текущего проекта или были созданы. Защищенные от записи библиотеки не отображаются. Чтобы добавить Ваш новый шаблон функции к одной из этих библиотек шаблонов, выберите подходящую запись и нажмите **OK**.

Выбор переменных как параметров

Вы можете определить переменные шаблона функции как параметры шаблона *(см. стр. 767)*. Чтобы определить переменные шаблона функции как параметры шаблона, выполните следующее:

Шаг	Действие
1	В диалоговом окне **Save as Function Template** щелкните кнопку **...** справа от текстового поля **Parameters**. **Результат**: Отображается диалоговое окно **Select Variables as Parameters**. Оно отображает переменные, которые определены в выбранном приложении.
2	Установите флажок индивидуальной переменной, чтобы выбрать его как параметр шаблона для шаблона функции. Или установите флажок корневого узла, чтобы выбрать все элементы ниже этого узла.
3	Щелкните по кнопке **OK**. **Результат**: выбранные переменные отображаются в текстовом поле **Parameters** диалогового окна **Save as Function Template**. Они отображаются в таблице **Parameters** диалогового окна **Add Function From Template**, где Вы можете присвоить для этих параметров **New Value**s (Новые Значения).

Перезапись существующего шаблона функции

Чтобы перезаписать существующий шаблон функции с выбранной прикладной функцией, выполните следующее:

Шаг	Действие
1	В диалоговом окне **Save as Function Template** щелкните по кнопке **...** справа от текстового поля **Template Name**.
2	Найдите уже доступный шаблон функции, который Вы хотите заменить.
3	Выберите шаблон функции, который Вы хотите заменить. **Результат**: имя этого шаблона функции вставлено в текстовом поле **Template Name** и имя библиотеки шаблонов, где оно сохранено, вставлено в текстовое поле **Template Library**.
4	Нажмите **OK**, чтобы закрыть диалоговое окно **Save as Function Template** и заменить выбранный шаблон функции новой прикладной функцией.

Добавление дополнительной информации к новому шаблону функции

Чтобы добавить дополнительную информацию к новому шаблону функции, нажмите кнопку ... справа от текстового поля **Properties**. Открывается диалоговое окно **Properties**. Оно позволяет Вам вводить дополнительную информацию для шаблона функции. Так как диалоговое окно идентично окну для шаблонов устройства и библиотек шаблонов, смотрите описание в разделе *Добавление дополнительной информации для шаблонов и библиотек шаблонов (см. стр. 739)*.

Часть IX

Поиск и устранение неисправностей и FAQ (часто задаваемые вопросы)

Содержание этой части

Эта часть содержит следующие главы:

Глава	Название главы	Страница
35	Дженерик - поиск и устранение неисправностей, часто задаваемые вопросы	775
36	Доступ к контроллерам - поиск и устранение неисправностей, часто задаваемые вопросы	785

Глава 35

Дженерик - поиск и устранение неисправностей, часто задаваемые вопросы

Раздел 35.1

Часто задаваемые вопросы

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Как я могу включить и сконфигурировать аналоговые входы в CANopen?	777
Почему выполнение запуска SoMachine иногда более медленное?	778
Как я могу управлять ярлыками и меню?	779
Как я могу увеличить предел памяти, доступный для SoMachine в 32-разрядных операционных системах?	781
Как я могу уменьшить потребление памяти SoMachine?	782
Как я могу увеличить производительность по времени построения в SoMachine?	782
Что я могу сделать в случае проблем с Modbus IOScanner на последовательной линии?	783
Что я могу сделать, если моя коммуникация по Network Variables List (NVL) была приостановлена?	784
Что я могу сделать, если многочисленная загрузка на контроллере HMI неуспешна?	784

Как я могу включить и сконфигурировать аналоговые входы в CANopen?

Обзор

Этот раздел содержит инструкции по включению аналоговых входов согласно стандарту CANopen установкой SDO (Service Data Object - Объект служебных данных) 6423 на значение 1.

Процедура

Шаг	Действие
1	Дважды щелкните по узлу своего аналогового устройства CANopen в **Devices tree**.
2	На вкладке **CANopen Remote Device** редактора включите опцию **Enable Expert Settings**. **Результат**: Отображены дополнительные вкладки, и вкладка **Service Data Object** заполнена информацией.
3	Откройте вкладку **Service Data Object** и нажмите кнопку **New...**. **Результат**: Отображено диалоговое окно **Select item from object directory**.
4	Из списка объектов выберите объект **6423**, введите в поле **Value** (Значение) 1 и нажмите **OK**. **Результат**: передача аналоговых входов по шине CANopen активирована. Теперь Вы можете сконфигурировать параметры аналоговых значений, как описано в руководстве аппаратными средствами Вашего устройства.

Почему выполнение запуска SoMachine иногда более медленное?

Обзор

Рядом с конфигурацией PC есть несколько других условий, которые могут увеличить время, используемое SoMachine во время запуска:

Фаза загрузки	Выполнение запуска
начальный запуск после установки SoMachine	При первом запуске после установки SoMachine, программное обеспечение генерирует свою рабочую среду в PC. Это делается только один раз, но оказывает значительное влияние на продолжительность первого запуска.
начальный запуск после перезагрузки	После перезагрузки PC время запуска SoMachine может быть более длительным, чем обычно, потому что Microsoft Windows использует некоторое время в фоновом режиме, чтобы запустить службы, необходимые, для работы SoMachine. Это может оказать влияние на продолжительность запуска и не может быть исключено.
последующие запуски	Пользователи испытывают лучшую производительность запуска, когда система была ранее запущена на PC.

Как я могу управлять ярлыками и меню?

Обзор

Меню и ярлыки программного обеспечения SoMachine отличаются в зависимости от текущего состояния, т.е. окна или редактора, который в настоящее время открыт.

Вы можете адаптировать ярлыки и меню к Вашим индивидуальным настройкам, или Вы можете загрузить стандартные ярлыки и меню SoMachine или CoDeSys, как описано в следующих разделах.

Настройка ярлыков и меню

Вы можете адаптировать ярлыки и меню к Вашим индивидуальным настройкам при помощи меню **Tools** → **Customize**.

Восстановление стандартных ярлыков и меню SoMachine

Чтобы восстановить стандартные ярлыки и меню SoMachine (после того, как Вы настраивали их), выполните следующее:

Шаг	Действие
1	Выполните команду **Customize** из меню **Tools**. **Результат**: Отображается диалоговое окно **Customize**.
2	В диалоговом окне **Customize** нажмите кнопку **Load...**. **Результат**: Отображается диалоговое окно **Load Menu**.
3	В диалоговом окне **Load Menu** переместитесь к папке...*Program Files\Schneider Electric\SoMachine Software\V4.0\LogicBuilder\Settings*, выберите файл *Standard.opt.menu* и нажмите **Open**. **Результат**: Диалоговое окно **Customize** теперь показывает стандартные параметры настройки SoMachine.
4	Чтобы загрузить эти стандартные параметры настройки в графический интерфейс пользователя SoMachine, нажмите **OK**.

Установка стандартных ярлыков и меню CoDeSys

Чтобы импортировать ярлыки и меню CoDeSys в Ваш графический интерфейс пользователя SoMachine, выполните следующее:

Шаг	Действие
1	Выполните команду **Customize** из меню **Tools**. **Результат:** Отображается диалоговое окно **Customize**.
2	В диалоговом окне **Customize** нажмите кнопку **Load**. **Результат:** Отображается диалоговое окно **Load Menu**.
3	В диалоговом окне **Load Menu** переместитесь к папке...*\Program Files\Schneider Electric\SoMachine Software\V4.0\LogicBuilder\Settings\OriginalCoDeSys*, выберите файл *Standard.opt.menu* и нажмите **Open**. **Результат**: диалоговое окно **Customize** теперь показывает параметры настройки CoDeSys.
4	Для загрузки этих параметров настройки CoDeSys в графический интерфейс пользователя SoMachine, нажмите **OK**.

ПРИМЕЧАНИЕ: Меню и ярлыки программного обеспечения SoMachine отличаются, в зависимости от окна или редактора, который в настоящее время открыт.

Расширение меню

Главные меню SoMachine и контекстные меню могут быть выведены на экран в свёрнутом или полном представлении. В свёрнутом режиме скрыты редко используемые или отключенные команды. После щелчка по пункту меню стрелки ▼ в нижней части меню соответствующее меню расширяется, показывая все его пункты меню.

Для того, чтобы всегда показать меню в полном режиме просмотра, активируйте опцию **Always show full menus** в диалоговом окне **Tools** → **Options** → **Features**.

Как я могу увеличить предел памяти, доступный для SoMachine в 32-разрядных операционных системах?

Обзор

Большие проекты SoMachine могут придвигать 32-разрядную операционную систему к техническому пределу относительно потребления памяти. Это происходит из-за того, что 32-разрядные операционные системы обеспечивают только 2 Гбайт памяти для пользовательских процессов, таких как SoMachine.

Идентификация большого проекта SoMachine

Проекты SoMachine могут считаться большими, если они содержат большое общее количество объектов (**Objects)**. **Objects**, которые доступны в проекте, такие как **Devices**, **POUs**, **Actions**, **DUTs**, **Global Variable Lists**, **Visualizations**, перечислены на вкладке **Statistics** диалогового окна **Project Information** (см. *SoMachine, Menu Commands, Online Help*). Однако это не просто общее количество **Objects**, которые могут указывать на крупный проект. Даже отдельные **Objects** по сути могут быть большими.

Включение переключателя на 3 Гбайт на 32-разрядных операционных системах Windows 7

Чтобы увеличить предел памяти для пользовательских процессов на Windows 7 32-разрядных операционных системах, Вы можете включить функцию переключателя на 3 Гбайт следующим образом:

Шаг	Действие
1	Перейдите в **Start Menu** → **All Programs** → **Accessories**.
2	Щелкните правой кнопкой по **Command Prompt** и выполните команду **Run as Administrator**.
3	Введите `bcdedit /set IncreaseUserVa 3072`.
4	Перезапустите компьютер.

Отключение 3 Гбайт переключателя 32-разрядных операционных систем Windows 7

Чтобы отключить увеличиение предела памяти для пользовательских процессов на 32-разрядных операционных системах Windows 7, Вы можете отлючить функцию переключателя на 3 Гбайт следующим образом:

Шаг	Действие
1	Перейдите в **Start Menu** → **All Programs** → **Accessories**.
2	Щелкните правой кнопкой по **Command Prompt** и выполните команду **Run as Administrator**.
3	Введите `bcdedit /deletevalue IncreaseUserVa`.
4	Перезапустите компьютер.

Как я могу уменьшить потребление памяти SoMachine?

Обзор

Этот раздел обеспечивает подсказки, которые могут помочь уменьшать потребление памяти процесса SoMachine в Вашей системе.

Разделение Вашего проекта SoMachine

Если Ваш проект SoMachine состоит из нескольких независимых частей, например, независимые корневые устройства (контроллер или панель), Вы можете разделить проект и создать независимые проекты SoMachine для каждого корневого устройства. Эти меньшие проекты SoMachine тогда требуют каждый меньшего количества памяти.

Закрытие редакторов SoMachine

Закрывайте каждый редактор SoMachine после того, как Вы установили соответствующие настройки, потому что каждый открытый редактор использует пространство памяти.

Как я могу увеличить производительность по времени построения в SoMachine?

Методы наиболее успешной практики

Список методов наиболее успешной практики может помочь Вам избежать медленной производительности при работе с SoMachine:

- Проверьте, что аппаратные средства PC удовлетворяют требованиям системы *(см. SoMachine, Введение)*.
- Используйте твердотельный накопитель (Solid State Drive - SSD) и проверьте, что доступно достаточное пространство памяти. Свяжитесь со своим администратором IT для получения дальнейшей информации.
- Рассмотрите удаление компонентов, в которых Вы не нуждаетесь, с помощью менеджера конфигурации SoMachine.
- Если Вы используете Vijeo- Designer, интегрированный в SoMachine, рассмотрите запрет функции автоматического экспорта символов. Чтобы достигнуть этого, активируйте опцию **Disable automatic symbol export** в диалоговом окне Logic Builder **Options** → **Vijeo-Designer** *(см. стр. 482)*.
- Рассмотрите активацию опции **Disable undo after deleting a DTM (performance optimization)** (Запрета отмены после удаления DTM (оптимизация производительности)) в диалоговом окне SoMachine Logic Builder **Options** → **FDT Options** (см. *SoMachine, Menu Commands, Online Help*), если эта функция используется не часто.
- Избегайте больших POU. У SoMachine нет предела для размера POU, запрограммированной на графических языках, таких, как LD. Тем не менее, наиболее успешная практика - это вызывать POU в последовательностях вместо того, чтобы создать одну большую POU.

Что я могу сделать в случае проблем с Modbus IOScanner на последовательной линии?

Обзор

Этот раздел содержит инструкции, которые могут помочь Вам решать проблемы, обнаруженные при использовании Modbus IOScanner на последовательной линии.

Состояние исключения приложения

Modbus IOScanner на последовательной линии сконфигурирован на Вашем контроллере, и одно из ведомых устройств Modbus отключено.

Если приложение переходит к состоянию исключения после загрузки или после перезагрузки контроллера, выполните следующее:

Шаг	Действие
1	Проверьте целостность своего кабеля.
2	Проверьте, что Ваш кабель правильно подключен между контроллером и последовательным ведомым Modbus.
3	Перезагрузите свой контроллер.

Ошибка, обнаруженная во время использования Modbus IOScanner на последовательной линии

Если…	Тогда…
обнаружена ошибка при использовании Modbus IOScanner на последовательной линии	флаг `xError`, связанный со слейвом, который был обнаружен как генерирующий, или вызывающий ошибку, установлен в TRUE.
	коммуникация НЕ остановлена (контроллер все еще пытается соединиться со слейвом).
	параметр `uiNumberOfCommunicatingSlaves` уменьшается, и `xAllSlavesOK` установлен в FALSE.

После того, как коммуникация со слейвом была восстановлена, нарастающий фронт требуется на записи xReset слейва:

- Для сброса xError.
- Для обновления значения `uiNumberOfCommunicatingSlaves`.
- Для обновления значения `xAllSlavesOK`.

Что я могу сделать, если коммуникация по моему Network Variables List (NVL) была приостановлена?

Проблема

Коммуникация NVL была приостановлена после онлайнового изменения.

Решение

Перезапустите целевой контроллер.

Что я могу сделать, если многочисленная загрузка неуспешна на контроллере HMI?

Проблема

Многочисленная загрузка не завершена успешно на контроллере HMI с устаревшим фирменным программным обеспечением.

Решение

Перезапустите многочисленную загрузку или запустите **Runtime Installer** выполнения в SoMachine Central через **Tool Access Bar** → **Maintenance** → **Download Firmware HMI**.

Глава 36

Доступ к контроллерам - поиск и устранение неисправностей, часто задаваемые вопросы

Содержание этой главы

Эта глава содержит следующие разделы:

Раздел	Тема	Страница
36.1	Поиск и устранение неисправностей: доступ к новым контроллерам	786
36.2	Часто задаваемый вопрос - что я могу сделать в случае проблем соединения с контроллером?	791

Раздел 36.1

Поиск и устранение неисправностей: доступ к новым контроллерам

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Доступ к новым контроллерам	787
Соединение через IP-адрес и адресную информацию	789

Доступ к новым контроллерам

Обзор

Чтобы получить доступ к новому контроллеру, адаптируйте сетевые параметры настройки контроллера к сети Вашего SoMachine PC. Этот раздел показывает пример по шагам.

Пример

Этот пример показывает шаги, для получения доступа к LMC058 с IP-адресом 192.168.1.33 от PC с IP-адресом 192.168.1.10, находящимся в подсети 255.255.255.0.

Шаг	Действие
1	Подключите контроллер непосредственно к PC, выполняющему SoMachine, или к сети PC, используя кабель Ethernet.
2	В SoMachine откройте вид **Controller selection** (выбора контроллера) редактора устройств *(см. стр. 97)*. **Результат**: контроллер LMC058 будет включен в список.
3	Чтобы адаптировать коммуникационные параметры настройки контроллера, щелкните правой кнопкой по контроллеру в списке **Controller selection** и выполните команду **Process communication settings...** из контекстного меню. **Результат**: диалоговое окно **Process communication settings** открывается.
4	В диалоговом окне **Process communication settings** введите свободный **IP-адрес**, доступый в Вашей сети. При конфигурировании IP-адресов примите во внимание сообщение об опасности, приведенное ниже.
5	Нажмите **OK**, чтобы подтвердить диалоговое окно **Process communication settings**.
6	В виде **Controller selection** соединитесь с контроллером.

Тщательно управляйте IP-адресами, потому что каждое устройство в сети требует уникального адреса. Наличие нескольких устройств с одним и тем же IP-адресом может вызвать непредусмотренное действие в Вашей сети и связанном с ней оборудовании.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННОЕ ДЕЙСТВИЕ ОБОРУДОВАНИЯ

- Проверьте, что у всех устройств есть уникальные адреса.
- Получите свой IP-адрес от Вашего системного администратора.
- Подтвердите, что IP-адрес устройства уникален прежде, чем перевести систему в работу.
- Не присваивайте один и тот же IP-адрес никакому другому оборудованию в сети.
- Обновите IP-адрес после клонирования любого приложения, которое включает коммуникации Ethernet с уникальным адресом.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

ПРИМЕЧАНИЕ: Некоторые контроллеры поддерживают параметр, который помогает препятствовать тому, чтобы они могли быть удаленно доступны (параметр **RemoteCommunicationAccess** контроллеров LMC •0•C.

Соединение через IP-адрес и адресную информацию

Обзор

Используемый протокол связи предлагает механизм для соединения с контроллером, независимый от типа соединения. Например, предоставляет доступ к целевому контроллеру, который подключен через Ethernet к другому транзитному контроллеру, который подключен через USB к самому PC.

1 PC
2 транзитный контроллер
3 целевой контроллер
4 USB
5 Ethernet

Адресная информация

В примере USB использует различный протокол. Поэтому обычно невозможно использовать IP-адрес, чтобы адресовать целевой контроллер. Вместо этого используется маршрутная информация, которая описывает путь соединения с целевым контроллером через 1 или более транзитных участков.

Эта маршрутная информация выведена на экран как подсказка записи списка контроллера (в следующем примере **[Address] 0000.2001**):

ПРИМЕЧАНИЕ: Так как этот адрес описывает только путь подключения контроллера, он может измениться после каждой модификации локальных PC или параметров настройки сетевого адаптера транзитного контроллера. Например, после активации или деактивации сетевых адаптеров или на запуск/остановку служб, которые используют сетевые адаптеры. Адрес определенной цели может отличаться от различных PC отправки.

Nodename

Так как **Nodename** (имя узла) контроллера - стабильный идентификатор в системе, он используется, чтобы идентифицировать цель.

Если IP-адрес выбран как **Connection Mode** (режим соединения**)**, его пробуют, чтобы получить информацию от самого **Nodename**. Некоторые контроллеры (такие как LMC •0•C) создают **Nodename** автоматически включая IP-адрес. Вы можете также сконфигурировать **Nodename** сами (как описано в FAQ: Почему контроллер не перечислен в виде выбора контроллера? *(см. стр. 793)*), позволяя системе найти контроллер со своим IP-адресом. Если IP-адрес отсутствует в nodename, пробуют получить IP-адрес от контроллера. Но не все устройства или их текущая версия фирменного программного обеспечения поддерживают сервис. В этом случае используйте **Connection Mode Nodename**, чтобы соединить или установить имя устройства, которое включает IP-адрес в скобках. Например, **MyDevice (192.168.1.30)**.

Раздел 36.2

Часто задаваемый вопрос - что я могу сделать в случае проблем соединения с контроллером?

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Часто задаваемый вопрос - Почему соединение с Контроллером невозможно?	792
Часто задаваемый вопрос - Почему имеющаяся связь между PC и контроллером прервана?	795

Часто задаваемый вопрос - Почему соединение с контроллером невозможно?

Почему соединение с контроллером невозможно даже при том, что IP-адрес, кажется, соответствует?

Если Вы установили IP-адрес контроллера, как описано в главе *Доступ к новым контроллерам (см. стр. 787)*, и Вы все еще не можете соединиться с контроллером, причиной может быть маска подсети. Так как используемый протокол связи требует идентичной маски подсети и на сайте отправителя, и на сайте получателя, может быть, что пинг (ping) к контроллеру успешен, но соединение не может быть установлено.

Чтобы решить эту проблему, выполните следующее:

Шаг	Действие
1	В SoMachine откройте вид **Controller selection** редактора устройства *(см. стр. 97)*.
2	Чтобы адаптировать коммуникационные параметры настройки контроллера, щелкните правой кнопкой по списку **Controller selection** и выполните команду **Process communication settings...** из контекстного меню. **Результат**: открывается диалоговое окно **Process communication settings**.
3	Адаптируйте маску подсети (**Subnet mask)**, сконфигурированную для контроллера точно к маске подсети Вашего SoMachine PC. **Пример:** Измените **255.255.0.0** на **255.255.255.0**.

ПРИМЕЧАНИЕ: После того, как Вы изменили **Connection Mode** в диалоговом окне **Controller selection**, может потребоваться выполнить процедуру входа в систему дважды для получения доступа к выбранному контроллеру.

Почему контроллер не перечислен в виде Communication Settings?

Если Вы устанавливаете соединение между контроллером и SoMachine PC при помощи активного пути, то представление **Communication Settings** выведено на экран в редакторе устройства *(см. стр. 111)*. Это - настройка по умолчанию для SoMachine V3.1 и более ранних версий.

Если контроллер по Вашему выбору не выведен на экран в представлении **Communication Settings**, Вы можете временно переключиться на установление соединения через IP-адрес следующим образом:

Шаг	Действие
1	Откройте диалоговое окно **Project Settings** → **Communication settings** из меню **Project**.
2	Выберите опцию **Dial up via “IP-Address** и подтвердите Ваши установки, нажав **OK**.
3	Адаптируйте сетевые параметры настройки контроллера к сети Вашего SoMachine PC, как описано в главе *Доступ е новым контроллерам (см. стр. 787)*.
4	Верните установление соединения назад к активному пути. Контроллер должен теперь быть выведен на экран в виде **Communication Settings**

Почему контроллер не перечислен в представлении выбора контроллера?

Если Вы не находите свой контроллер в списке вида **Controller selection** редактора устройств *(см. стр. 97)*, причина может состоять в том, что двум различным устройствам присвоен тот же **Nodename**. Если двум устройствам присвоен тот же **Nodename**, только 1 из этих устройств перечислено в списке **Controller selection**.

Вы должны тщательно управлять **Nodename**, потому что каждое устройство в сети требует уникального **Nodename**. Наличие многих устройств с тем же **Nodename** может вызвать непредсказуемую работу Вашей сети и связанного с ней оборудования.

⚠ ПРЕДУПРЕЖДЕНИЕ

НЕПРЕДНАМЕРЕННОЕ ДЕЙСТВИЕ ОБОРУДОВАНИЯ

- Прежде, чем поместить систему в работу, проверьте, что у всех устройств есть уникальные **Nodenames**.
- Обновите **Nodename** после клонирования любого приложения, которое включает коммуникации Ethernet с уникальным **Nodename**.

Отказ следовать этим инструкциям может привести к смерти, серьезной травме или повреждению оборудования.

Чтобы изменить **Nodename** устройства, выполните следующее:

Шаг	Действие
1	Щелкните правой кнопкой по устройству, которому присвоено дублированное **Nodename** в списке **Controller selection**, и выполните команду **Change device name** из контекстного меню. **Результат**: Отображается диалоговое окно **Change device name**.
2	В диалоговом окне **Change device name** введите уникальное **Nodename** в текстовое поле **New**.
3	Нажмите **OK**, чтобы подтвердить и закрыть диалоговое окно **Change device name**.

Шаг	Действие
4	В виде **Controller selection** нажмите кнопку **Update**, чтобы обновить список устройств. **Результат**: Второе устройство с тем же **Nodename** устройства, которое Вы просто изменили, будет теперь отображено в списке.
5	Повторите шаги 1... 4, пока Вы не устранили какие-либо дублированные **Nodenames**.

ПРИМЕЧАНИЕ: Некоторые контроллеры, такие как LMC •0•C контроллеры, создают **Nodename** автоматически из имени устройства проекта после загрузки проекта и IP-адреса (например, **MyLMC (192.168.1.30)**). Это автоматическое имя перезаписывает **Nodename**, которое Вы присвоили, если какие-либо изменения выполняются на контроллере.

Часто задаваемый вопрос - Почему имеющаяся связь между PC и контроллером прервана?

Почему имеющаяся связь между PC и контроллером прервана?

Может быть, необходимо перезапустить шлюз следующим образом:

Шаг	Действие
1	Щелкните правой кнопкой по иконке **Gateway Tray Application** в панели задач Windows
2	Выполните команду **Restart Gateway** из контекстного меню.

Приложения

Что находится в этом приложении?

Приложение содержит следующие главы:

Глава	Название главы	Страница
A	Сетевые коммуникации	799
B	Использование сервера OPC 3	807
C	Язык сценария	821
D	Управление пользователями для Soft PLC	869
E	Наборы функций контроллера для миграции	879

Приложение A

Сетевые коммуникации

Содержание этой главы

Эта глава содержит следующие темы:

Тема	Страница
Топология сети	800
Адресация и маршрутизация	801
Структура адресов	803

Топология сети

Обзор

Сеть управления SoMachine - система, запрограммированная на конфигурирование себя (присвоение адресов), чтобы поддерживать прозрачную коммуникационную среду и направлять пакеты между различными сетями. Механизм маршрутизации достаточно прост, так что любой узел в сети, т.е. даже узлы с низкими ресурсами, в состоянии перенаправить пакеты. Так избегают больших таблиц маршрутизации, сложных вычислений или запросов в течение времени выполнения.

Сеть управления сконфигурирована иерархически, т.е. у каждого узла есть один родительский узел и произвольное число дочерних элементов. Узел без родителя упоминается как узел верхнего уровня. Циклы не разрешены, т.е. сеть управления имеет древовидная структура.

Отношения родителей и детей являются результатом спецификации сегментов сети. Сегмент сети соответствует, например, локальному Ethernet или последовательному двухточечному соединению. Различают основную сеть (mainnet) и подсети (subnet). У каждого узла есть, самое большее, одна основная сеть, где он предполагает своего родителя. Для каждого узла может быть сконфигурировано произвольное число подсетей. Узел действует для всех них как родитель.

Если бы сегмент сети был определен одновременно как подсеть нескольких узлов, у сети было бы несколько родителей. Однако получающаяся конфигурация будет инвалидной, поскольку каждому сегменту сети позволено иметь только одного родителя-одиночку.

Адресация и маршрутизация

Обзор

Адресация отображает топологию сети управления к уникальным адресам. Адрес узла *(см. стр. 804)* создан иерархически.

Для каждого сетевого соединения локальный адрес, идентифицирующий узел исключительно в его соответствующей локальной сети, выделен соответствующим блочным драйвером. Для всего адреса узла этому локальному адресу предшествует индекс подсети, которому присвоена локальная сеть родителем.

Кроме того, этому должен предшествовать адрес узла его родителя.

Длина индекса подсети (в битах) определена устройством, тогда как длина локального адреса определена типом сети.

Узел без основной сети - узел верхнего уровня с адресом 0. Узел с основной сетью, которая не содержит родителя, является также узлом верхнего уровня и будет присвоен его локальному адресу в основной сети.

Пример: Основная сеть и подсети

В примере адреса дочерних узлов даны в шестнадцатеричном представлении. Первые 4 цифры представляют адрес определенного родителя в основной сети. Например, 0x007A=122 для PLC1. Следующий байт (выведенный синим) зарезервирован для индекса подсети, и он сопровождается локальным адресом, например, C=12 для узла ID 12.

Из-за структурирования адреса, алгоритм маршрутизации может быть сохранен относительно экономным. Например, нет необходимости в каких-либо таблицах маршрутизации. Информация запрашивается локально: на собственном адресе и на адресе родительского узла.

Вслед за этим узел может правильно обработать пакеты данных.

- Если целевой адрес равняется адресу текущего узла, он определяется как получатель.
- Если целевой адрес начинается с адреса текущего узла, пакет предназначен для дочернего элемента или потомка узла и должен быть передан.
- Иначе, получатель не потомок текущего узла. Пакет должен быть передан к собственному родителю.

Относительная адресация

Относительная адресация - специальная функция. Относительные адреса *(см. стр. 805)* не содержат номер узла получателя, но непосредственно описывают путь от отправителя до получателя. Принцип подобен относительному пути в файловой системе: адрес состоит из количества шагов, на которые пакет должен переместиться вверх, т.е. на следующего соответствующего родителя и последующий путь вниз к целевому узлу.

Преимущество относительной адресации состоит в том, что 2 узла в том же поддереве в состоянии продолжать коммуникацию, когда все поддерево перемещено в другую позицию в полной сети управления. В то время как абсолютные адреса узлов изменятся из-за такого перемещения, относительные адреса все еще допустимы.

Определение адресов

Узел пытается определить свой собственный адрес в то время как это приходит от его родителя или является ли он сам узлом верхнего уровня. С этой целью узел отправляет определение адреса с помощью широковещательного сообщения к его основной сети во время начальной загрузки. Пока на это сообщение не отвечают, узел считает себя узлом верхнего уровня, хотя он будет продолжать пытаться обнаружить родительский узел. Родительский узел ответит уведомлением об адресе. Вслед за тем узел завершит свой собственный адрес и передаст его к подсетям.

Определение адреса может быть выполнено при начальной загрузке или по запросу PC программирования.

Структура адресов

Обзор

Ниже приведено подробное описание структуры следующих типов адресов:

- Сетевые адреса *(см. стр. 803)*
- Адреса узлов *(см. стр. 804)*
- Абсолютные и относительные адреса *(см. стр.805)*
- Широковещательные адреса *(см. стр. 806)*

Сетевой адрес

Сетевые адреса представляют отображение адресов сетевого типа (например, IP-адреса) на логические адреса в сети управления. Это отображение обрабатывается соответствующим блочным драйвером. В Ethernet с IP-адресами класса C, первые 3 байта IP-адреса одни и те же для всех сетевых устройств. Поэтому последние 8 битов IP-адреса достаточны как сетевой адрес, так как они позволяют однозначное отображение между 2 адресами в блочном драйвере.

Узел имеет отдельные сетевые адреса для каждого сетевого соединения. Различные сетевые соединения может иметь один и тот же сетевой адрес, так как этот адрес должен быть уникальным только локально для каждого сетевого соединения.

Терминология: В целом сетевой адрес узла без утверждения сетевого соединения относится к сетевому адресу в основной сети.

Длина сетевого адреса определена в битах и может быть выбрана блочным драйвером как требуется. В сегменте сети одинаковая длина должна использоваться для всех узлов.

Сетевой адрес представлен как массив байтов со следующим кодированием:

- Длина сетевого адреса: n бит
- Требуемые байты: b = (n + 7) DIV 8
- (n MOD 8) самые низкоуровневые биты первого байта и все остающиеся (n DIV 8) байты используются для сетевого адреса.

Пример – Сетевой аадрес

Длина: 11 бит

Адрес: 111 1000 1100

Пример кодирования сетевого адреса

Byte	0								1							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
						1	1	1	1	0	0	0	1	1	0	0

Reserved (0)

Адрес узла

Адрес узла указывает абсолютный адрес узла в сети управления, и поэтому уникален в целом дереве. Адрес состоит максимум из 15 компонентов адреса, каждый состоящий из 2 байтов. Чем ниже узел расположен в сетевой иерархии, тем длиннее его адрес.

Адрес узла состоит из частичных адресов всех предшественников узла и самого узла. Каждый частичный адрес состоит из одного или нескольких компонентов адреса. Поэтому длина - всегда число кратное 2. Частичный адрес узла сформирован из сетевого адреса узла в его основной сети и индексе подсети основной сети в родительском узле. Биты, требуемые для индекса подсети, определяются маршрутизатором родительского узла. Биты заполнителя вставлены между индексом подсети и сетевым адресом, чтобы гарантировать, что длина частичного адреса - число кратное 2 байтам.

Особые случаи:

- У узла нет основной сети: Это означает, что нет никакого индекса подсети, ни сетевого адреса в основной сети. В этом случае адрес установлен в 0x0000.
- Узел с основной сетью, но без родителя: В этом случае принят индекс подсети с 0-разрядной длиной. Частичный адрес соответствует сетевому адресу, добавленному битами заполнителя при необходимости.

Пример – адрес узла

274 = 0274. To improve readability, the output should always include the leading zeros.

Представление адреса узла всегда шестнадцатеричное. Отдельные компоненты адреса (2 байта в каждом случае) разделены двоеточием (:). Байты внутри компонента отображаются последовательно без разделителя (см. пример выше). Так как это представляет массив байтов и не 16-битовое значение, компоненты не отображаются в формате с прямым порядком байтов. Для вручную введенных адресов недостающие цифры в компоненте адреса заполнены начальными нулями слева: 274 =0274. Чтобы улучшить удобочитаемость, вывод должен всегда включать начальные нули.

Абсолютные и относительные адреса

Коммуникация между 2 узлами может быть основана на относительных или абсолютных адресах. Абсолютные адреса идентичны адресам узлов. Относительные адреса специфицируют путь от отправителя к получателю. Он состоит из смещения адреса и пути спуска к получателю.

(Отрицательное) смещение адреса указывает число компонентов адреса, на которое пакет должен быть передан вверх в дереве, прежде чем он сможет быть передан снова вниз от общего родителя. Так как узлы могут использовать частичные адреса, состоящие больше чем из одного компонента адреса, количество родительских узлов, которые будут переданы, всегда = смещению адреса. Это означает, что разграничение между родительскими узлами больше не однозначно. Поэтому общая начальная часть адресов коммуникационных партнеров используется в качестве родительского адреса. Каждый компонент адреса считается как восходящий шаг, независимо от фактических родительских узлов. Любая ошибка, представленная этими предположениями, может быть обнаружена соответствующим родительским узлом и должна быть обработана узлом правильно.

По прибытии к общему родителю относительный путь (массив компонентов адреса) тогда следует вниз нормальным способом.

Формально: адрес узла получателя формируется удалением последних компонентов смещения адреса из адреса узла отправителя и добавлением относительного пути к остающемуся адресу.

Пример

В примере буква будет представлять компонент адреса, тогда как точка разделит определенные узлы. Так как узлу позволяется иметь много адресных компонентов, позволено иметь много букв в примере.

Узел A: a.bc.d.ef.g

Узел B: a.bc.i.j.kl.m

- Адрес самого низкого общего родителя: a.bc
- Относительный адрес от A до B: -4/i.j.kl.m (число -4 следует из 4 компонентов d, e, f, и g. Поэтому пакет должен быть поднят).

Относительный адрес должен быть скорректирован с каждым проходом через промежуточный узел. Достаточно скорректировать смещение адреса. Это всегда делается родительским узлом: Если узел получает пакет от одной из его подсетей, смещение адреса увеличено длиной компонента адреса этой подсети.

- Если новое смещение адреса < 0, пакет должен быть передан к родительскому узлу.
- Если смещение адреса 1 ≥ 0, пакет должен быть передан к дочернему узлу локального адреса, который расположен в позиции, описанной смещением адреса в относительном адресе. Во-первых, смещение адреса должно быть увеличено длиной локального адреса дочернего узла, чтобы гарантировать, что узел видит корректный адрес.

Специальная ситуация возникает, когда ошибка, описанная выше, происходит при определении общего родителя. В этом случае смещение адреса в “реальном” общем родителе отрицательно, но величина больше, чем длина частичного адреса подсети, из которой происходит пакет. Узел должен обнаружить этот случай, вычислить локальный адрес следующего дочернего узла на основе адреса предыдущего узла и различия в длине, и адаптировать смещение адреса таким образом, что следующий узел будет видеть корректный относительный адрес. Кроме того, сами компоненты адреса остаются неизменными, и только смещение адреса будет изменено.

Широковещательные адреса

Есть 2 типа широковещательных сообщений - глобальные и локальные. Глобальная широковещательная передача отправляется во все узлы в сети управления. Пустой адрес узла (длина 0) зарезервирован для этой цели.

Локальные широковещательные сообщения отправляются во все устройства сегмента сети. С этой целью все биты сетевого адреса установлены в 1. Это возможно и в относительных и в абсолютных адресах.

Блочный драйвер должен быть в состоянии обработать оба широковещательных адреса, т.е. пустые сетевые адреса и сетевые адреса со всеми битами, установленными в 1, должны быть интерпретированы и отправлены как широковещательные.

Приложение B

Использование сервера OPC 3

Обзор

Описание, представленное в этой главе, предназначено людям, опытным в технологии OPC сервера.

Файл *OPC_V3_how_to_use_E.pdf*, который автоматически установлен с SoMachine в каталоге *C:\Program Files\Schneider Electric\SoMachine OPCServer*, предоставляет более подробное описание того, как сконфигурировать OPC сервер.

Содержание этой главы

Эта глава содержит следующие темы:

Тема	Страница
Общая информация	808
Объявление переменной, используемой с OPC	810
Конфигурация OPC сервера	813
Использование CoDeSys OPC сервера	819

Общая информация

Что такое OPC?

OPC - стандартизированный интерфейс для доступа к обработке данных. Он основывается на стандарте Microsoft COM/DCOM2 (Component Object Model / Distributed COM), который был расширен из-за требований доступа к данным в автоматизации, где интерфейс используется, главным образом, для считывания данных из или записи данных в контроллер.

Примеры типичных OPC клиентов

- визуализации
- программы, цель которых состоит в сборе эксплуатационныхе данных

Примеры типичных провайдеров OPC серверов

- системы управления
- интерфейсные платы полевых шин

Что такое OPC сервер?

OPC сервер - исполняемая программа, которая запускается автоматически во время установления соединения между клиентом и контроллером. Следовательно, OPC сервер в состоянии сообщить клиенту об измененных значениях переменных или состояниях.

OPC сервер обеспечивает все переменные (называемые **Items** в OPC), которые доступны на контроллере (**Item Pool** или **Address Space**). Этими элементами управляют в **Data Cache**, который помогает гарантировать быстрый доступ к их значениям. Также возможен прямой, не кэшируемый доступ к переменным контроллера.

В OPC сервере элементы могут быть организованы в так называемые **Groups** (группы **Private** и **Public**).

Частные (private) группы могут быть составлены в клиенте произвольно из определенных элементов. Первоначально они не производят воздействия на группировки в сервере OPC, но при необходимости могут быть преобразованы к Public Groups. Работа с Private Groups, например, полезна, чтобы быть в состоянии активировать или деактивировать определенные группы переменных всего одной единственной командой, в зависимости от того, должны ли они быть доступными или нет.

Сгруппированные данные должны быть считаны из сервера OPC когерентно, т.е. все переменные должны быть считаны одновременно. Однако это не всегда возможно в случае целевых систем с ограниченными коммуникационными буферами.

Из-за характеристик COM / DCOM можно получить доступ к OPC, работающему на другом компьютере. Также допускается, чтобы одновременно больше чем один клиент получил доступ к источнику данных. Применимость различных языков (C++, Visual Basic, Delphi, Java) является другим преимуществом концепции.

Обзор сервера CoDeSys OPC Server 3

CoDeSys OPC сервер основывается на PLCHandler фирмы 3S - Smart Software Solutions GmbH. Этот коммуникационный модуль разрешает непосредственную связь к тем контроллерам, которые программируемы в CoDeSys.

Сервер OPC V.3 или более поздних версий поддержвает следующие спецификации OPC:

- OPC Common Definitions and Interfaces Version 1.0
- Data Access Custom Interface Standard Version 1.0
- Data Access Custom Interface Standard Version 2.05A
- Data Access Custom Interface Standard Version 3.0
- Data Access Automation Interface Standard Version 2.0

Связь между сервером OPC и контроллером может быть выполнена через следующий интерфейс:

- Gateway V3

Декларирование переменной, которая будет использоваться с OPC

Шаги для объявления переменной

Объявляйте переменную, которая используется с OPC, следующим образом:

Шаг	Действие	Пример
1	Создайте проект.	**ExampleOPC**
2	Добавьте и выберите контроллер, используя команду **Add Device**.	–
3	Сконфигурируйте имя устройства, дважды щелкнув по узлу устройства, чтобы сделать его доступным для редактирования.	**MyPLCexample**
4	Создайте **PROGRAM** в своем приложении, щелкнув правой кнопкой по элементу **Application** и выполнив команду **Add Object** → **POU...**.	Пример программы: Инкрементируйте UINT переменную: VARIABLEEXAMPLE
5	Сконфигурируйте имя **PROGRAM**	**MyPOUexample**
6	Дважды щелкните по задаче и свяжите **PROGRAM** с задачей.	Пример задачи: **MAST** Пример **PROGRAM: MyPOUexample**
7	Выполните команду **Build All** из меню **Build** и проверьте, что нет никаких ошибок, обнаруженных во время выполнения команды **Build**. 	–

Шаг	Действие	Пример
8	Создайте объект **Symbol configuration** в своем приложении, щелкнув правой кнопкой по элементу **Application** и выполнив команду **Add Object** → **Symbol configuration...**. 	–
9	В диалоговом окне **Add Symbol configuration** нажмите **Open**.	–
10	Щелкните по ссылке **Refresh**.	–
11	Разверните элемент **Variables** в списке **Available variables**.	–
12	Выберите переменную, которую Вы хотите использовать совместно с Вашим клиентом OPC из Вашей программы.	Пример переменной: VARIABLEEXAMPLE Пример программы: **MyPOUexample**
13	Щелкните кнопку **>**, чтобы отправить переменную в базу совместно используемых данных, чтобы сделать её доступной для клиента OPC. 	–

Шаг	Действие	Пример
14	Выполните команду **Build All** из меню **Build** и проверьте, что нет никакой ошибки, обнаруженной во время выполнения команды **Build**	–
15	Закройте вкладку **Symbol configuration**.	**Комментарий**: В директории, где Вы сохранили свой проект, Вы найдете XML-файл, включающий список переменных, которые доступны для клиента OPC.
16	Соедините свой PC с Вашим контроллером при помощи вкладки **Communication Setting**.	–
17	Загрузите приложение.	–
18	Запустите приложение. 	–

XML-файл, перечисляющий переменные, доступные для клиента OPC

В каталоге, где Вы сохранили свой проект, автоматически создается XML-файл, который описывает список переменных, доступных для клиента OPC.

Переменные, отображенные на %I

Переменные, отображенные на %I, выбранные в конфигурации **Symbol**, доступны в клиенте OPC не автоматически.

Добавьте путь к переменной вручную.

Пример:

```
M251.Application.IOCONFIG_GLOBALS_MAPPING.temperatureLabo
```

Конфигурация OPC сервера

Запуск инструмента конфигурирования OPC

Сконфигурируйте OPC сервер и соедините его с проектом, который Вы создали, следующим образом:

Шаг	Действие
1	Перейдите к директории: *C:\Program Files\Schneider Electric\SoMachine OPCServer*
2	Дважды щелкните: файл *OPCConfig.exe*
3	Инструмент конфигурирования *OPCconfig.exe* позволяет генерировать файл INI, который необходим, чтобы инициализировать OPC сервер с желаемыми параметрами для связи между проектом CoDeSys и контроллером (контроллерами).

Инструмент конфигурирования OPC

Инструмент конфигурирования содержит следующие элементы:

- строка меню
- структурный вид для отображения присвоений одного или нескольких контроллеров к серверу
- диалоговое окно конфигурации, которое соответствует выбранной в настоящее время записи в древовидной структуре.

После запуска инструмента, он появится следующим образом, содержа общие настройки по умолчанию:

Меню File инструмента конфигурирования OPC

Меню **File** обеспечивает команды для загрузки и сохранения конфигурационных файлов в / из инструмента конфигурирования:

Команда	Комбинация клавиш	Описание
Open	CTRL+O	Для редактирования существующей конфигурации. Диалоговое окно для открытия файла по умолчанию открывается. Выберите уже существующий INI файл. Фильтр автоматически установлен на *OPCconfg Files *.ini.* Конфигурация, описанная в выбранном INI файле, будет загружена в инструмент конфигурирования.
New	CTRL+N	Для создания новой конфигурации. Если конфигурация будет открыта, Вас спросят, должна ли она быть сохранена прежде чем быть закрытой. Затем инструмент конфигурирования покажет настройки по умолчанию.
Save	CTRL+S	Сохраняет текущую конфигурацию в загруженный в настоящее время INI файл.
Save as	–	Сохраняет текущую конфигурацию в файл с другим именем, которое Вы можете определить в диалоговом окне по умолчанию.
<n> недавно открытых INI-files	–	Список файлов INI, которые были отредактированы, начиная с последнего раза запуска инструмента. Вы можете выбрать файл, чтобы перезагрузить его в инструмент конфигурирования.
Exit	–	Завершает действие инструмента. Если какие-либо изменения в текущей конфигурации не были сохранены, Вас попросят сделать это.

Меню редактирования Edit инструмента конфигурирования OPC

Меню **Edit** обеспечивает команды для редактирования дерева конфигурации в левой части конфигуратора.

Команда	Комбинация клавиш	Описание
New Redundancygroup	CTRL+G	Запись Redundant Group будет добавлена ниже **Server**. Если уже есть контроллер или Redundant Groups, перечисленные в дереве, новая Redundant Group будет добавлена в конце. По умолчанию новая запись именуют **Redundant<n>** с n - последовательным числом, начинающимся с 1. Чтобы переименовать запись, выберите его в дереве и/ или используйте команду **Edit** → **Rename PLC** или щелкните по нему дважды, чтобы сделать его доступным для редактирования.
Append PLC	CTRL+O	Запись контроллера будет добавлена ниже **Server**. Новый контроллер будет добавлен в конце существующего дерева. По умолчанию новую запись называют **PLC <n>** с n - последовательным числом, начинающимся с 1. Чтобы переименовать запись, выберите его в дереве и /или используйте команду **Edit** → **Rename PLC** или щелкните по нему дважды, чтобы сделать его доступным для редактирования.
Delete PLC	CTRL+D	Выбранная в настоящее время запись контроллера будет удалена из дерева конфигурации.
Rename PLC	CTRL+R	Выбранная в настоящее время запись контроллера может быть переименована.
Reset PLC	CTRL+Z	Параметры настройки выбранной в настоящее время записи контроллера будут сброшены к значениям по умолчанию, определенным в **PLC Default Settings**.
PLC Default Settings...	–	еще не доступны

Конфигурирование OPC сервера

Сконфигурируйте сервер OPC следующим образом:

Шаг	Действие
1	Щелкните правой кнопкой по значку **Server** и выполните команду **Append PLC**:
2	Выберите **GATEWAY3** из списка **Interface**.
3	Дважды щелкните по элементу **PLC1**, чтобы переименовать его (например: **MyPLCexample**).

Шаг	Действие
4	Дважды щелкните по значку **Connection** и нажмите кнопку **Edit**. **Результат**: Диалоговое окно **CoDeSys address of PLC** будет выведено на экран:
5	Чтобы быть в состоянии получить доступ к Вашей переменной от Вашего клиента OPC, введите адрес контроллера (например **MyPLCexample**). Адрес дан в диалоговом окне SoMachine **Communication Setting** Вашего проекта.

Адрес может быть физическим или логическим. Чтобы избежать переконфигурирования значения адреса, когда в Вашем проекте есть много устройств, Вы должны использовать логические адреса.

Логическая адресация

В нашем примере адрес: **MyPLCexample**.

Введите непосредственно **Node name**, данное на вкладке **Communication Setting, MyPLCexample** в Вашем проекте. Чтобы сконфигурировать **Node name**, нажмите кнопку **Edit**.

Физическая адресация

В нашем примере адрес: 007F.6200.0001

Продолжите следующим образом:

Шаг	Действие
1	Возьмите этот адрес и сконфигурируйте значение **CoDeSys address of PLC**:
2	Нажмите кнопку **OK** **Результат**: На экран будет выведено следующее диалоговое окно:

Шаг	Действие
3	Откройте меню **File** и выполните команду **Save As**. Щелкните по кнопке **OK**, и на экран будет выведено следующее диалоговое окно:
4	Выберите *OPCServer.ini* и нажмите **Save**. **ПРИМЕЧАНИЕ**: Обратите внимание на то, что имя файла должно быть *OPCServer.ini*. Не используйте другое имя файла.

Использование CoDeSys OPC сервера

Обзор

После установки OPC сервера, он должен быть предложен для выбора клиентом OPC (например, визуализацией). Имя OPC сервера - **CoDeSys. OPC.DA.**

OPC сервер будет запущен автоматически операционной системой, как только клиент установит соединение. OPC сервер завершится автоматически, как только клиенты закрыли свои соединения с сервером.

В панели задач не будет никакого значка сервера OPC. Он появится только в **Windows Task Manager** как процесс.

Выполнение клиента OPC на PC, не выполняющем SoMachine

Чтобы быть в состоянии выполнить OPC клиента на PC, где SoMachine не установлен, выполните следующее:

- Обратитесь к главе *Installation of the CoDeSys OPC Server* в SoMachine *Installation and Configuration Manager User Guide* и выполните следующие действия:
  - Установите шлюз на PC, где работает OPC клиент.
  - В зависимости от OPC клиента Вы должны запустить файл *WinCoDeSysOPC.exe*.
- Скопируйте файл *OPCServer.ini* в ту же директорию, где установлен *WinCoDeSysOPC.exe*.

Приложение C

Язык сценариев

Содержание этой главы

Эта глава содержит следующие разделы:

Раздел	Тема	Страница
C.1	Общая информация	822
C.2	Примеры механизма выполнения сценария Schneider Electric	833
C.3	Примеры механизма выполнения сценария CoDeSys	848

Раздел C.1

Общая информация

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Введение	823
Выполнение сценариев	826
Методы наиболее успешной практики	829
Чтение документации API.NET	830
Точки входа	832

Введение

Язык сценариев SoMachine

Язык сценариев SoMachine обеспечивает мощный инструмент для автомизации последовательностей. Вы можете запустить отдельные команды или сложные последовательности команд непосредственно от среды программы SoMachine или из командной строки Windows. Язык сценариев SoMachine - модульный язык на основе IronPython 2.7. Интерпретатор IronPython интегрирован в среду разработки SoMachine. Эта реализация позволяет пользоваться обширными библиотеками платформы Python. Среди прочего они обеспечивают доступ к файлам в сетях.

Эта глава не предоставляет пример для каждого члена механизма выполнения сценария (Script Engine), потому что это вышло бы за объем этого документа. Примеры, приведенные здесь, были выбраны из-за определенной логики, которая работает как общая нить через API (Application Programming Interface - интерфейс программирования приложения). Если Вы следуете за этим потоком, Вы можете найти дальнейших участников в документе *Automation Platform SDK*, который обеспечивается как онлайн-справка на веб-странице CoDeSys. Тогда Вы будете в состоянии использовать этих участников таким же образом, как описано для примеров в этой главе.

Среды программирования для Python

Доступны следующие свободные IDE (Integrated Development Environments - Интегрированные среды разработки):

IDE	Функции
Notepad++	Подсветка синтаксиса Универсальный редактор
IDLE (Integrated DeveLopment Environment)	Подсветка синтаксиса
Visual Studio plugin	Подсветка синтаксиса IntelliSense для конструкций языка IronPython (и.NET) Интегрированная функция отладки Простой в использовании Требует Visual Studio
PyCharm	Подсветка синтаксиса IntelliSense Интегрированная функция отладки Простой в использовании Автономный программный продукт Free Community Edition версия 3.0 и более поздние версии, поддерживающие многочисленные функции, такие как: Для чистого кодирования Python и изучения Умный редактор, с завершением кода, выделение ошибки на лету, автоисправлении, и т.д. Автоматизированные рефакторинги кода и богатые навигационные возможности Интегрированный отладчик и поддержка поблочного тестирования. Собственные VCS (Система управления версиями) интеграции Настраиваемый пользовательский интерфейс и привязки клавиш, с VIM (текстовый редактор Vi IMproved) доступной эмуляцией

Примечания по совместимости

Python - динамический язык, используемый для большого разнообразия целей с акцентом на чистый и выразительный код. Он позволяет максимальную гибкость для разработчика при поддержании удобочитаемости кода. IronPython приносит Python к .NET и предоставляет собственный доступ к платформе и классам.NET. Реализация интерпретатора IronPyhon основывается на версии 2.7 Python. Есть много бесплатных учебных руководств и систем онлайн-справки, доступных в Интернете.

ПРИМЕЧАНИЕ: Несовместимость версии с Python V3.x. Следует иметь в виду, что использование Python языка может быть обновлено, и если так, можно удалить старые, осуждаемые конструкции языка. Поэтому запишите свои сценарии прямо совместимым способом. Это включает, например, использование оператора

```
from___future___import print_function.
```

Для получения дополнительной информации обратитесь непосредственно к веб-сайтам

- *http://wiki.python.org/*
- *http://docs.python.org/*

Примеры новых функций:

```
from___future__import print_function
from__future__import division
# New Python print syntax
print('Hello World!')
# Division
# Python 2 return an integer and rounds off
# Python 3 returns a float print(17/3)
```

Соглашения кодирования

Чтобы согласовать и упростить работу различных программистов на том же проекте программирования, им целесообразно договариваться об общем стиле программирования. Schneider Electric и CoDeSys договорились о принятии *Style Guide for Python Code - Руководства по стилю для кода Python*. Любые новые сценарии должны также соответствовать этому стандарту.

Для получения дополнительной информации обращайтесь к *Style Guide for Python Code* на *http://www.python.org/dev/peps/pep-0008/*.

Полезные ссылки

Для получения дополнительной информации обратитесь к следующим веб-сайтам:

- Официальная веб-страница Python, обеспечивающая учебное руководство и ссылки языка на *http://docs.python.org/*.
- Официальный блог Python на *http://blog.python.org/*.
- *Руководство для новичка по Python* на *http://wiki.python.org/moin/BeginnersGuide*.
- Статья в Wikipedia *Python (programming language)* на *http://en.wikipedia.org/wiki/Python_(programming language)*.
- Официальный форум CoDeSys обеспечивающий примеры и полезную информацию на *http://forum.codesys.com/*.
- Интерпретатор IronPython на *http://ironpython.codeplex.com/*.
- PyTools (Плагин Visual Studio) на *http://pytools.codeplex.com/*.
- *Поваренная книга IronPython* на *http://www.ironpython.info/index.php/Contents*.
- Free Galileo Openbook (доступно только на немецком языке) на *http://openbook.galileocomputing.de/python/*.

Выполнение сценариев

Обзор

Вы можете выполнить файлы сценария (*filename.py*), содержащие последовательность команд сценария, из пользовательского интерфейса SoMachine.
Для получения дополнительной информации по рабочим сценариям формирующих пользовательский интерфейс SoMachine, обращайтесь к главе *Script-Related Commands (см.SoMachine, Menu Commands, Online Help)*.

Командные файлы

Часто используемые команды

Команда	Описание
- `REM` или `::`	Строка - комментарий и будет проигнорирована.
`cd`	Изменения на другую директорию.
`echo off`	Команды не будут отображаться. С целью предотвращения отображения отдельных команд вставьте символ `@` перед командой.
`echo`	Отображает строку или переменную на консоли программирования.
`set`	Объявляет переменную и присваивает значение этой переменной.
`>`	Пишет вывод в файл. Если файл уже существует, он будет перезаписан.
`>>`	Добавляет вывод к файлу. Если файл ещё не будет существовать, он будет создан.

Пример приложения:

```
@echo off
REM Go to the directory where SoMachine is installed
cd "<Replace this with the path to the Central.exe, for example, C:\Program Files (x86)\Schneider Electric\SoMachine Software\>"
REM Run Central.exe with no graphical user interface and the full path to the script
Central.exe --noui --runscript="<Replace this with the full file path where the script is stored, for example, D:\MyScripts\TestScript.py>"
pause
```

Консольное приложение C#

Выполнение сценария в приложении C# позволяет Вам редактировать сценарий динамично, прежде чем сценарий будет выполнен процессором. Кроме того, некоторые предыдущие шаги могут быть выполнены также в приложении C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;
namespace ExecuteScriptExample
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                ProcessStartInfo psi = new ProcessStartInfo();

                // Specify the name and the arguments you want to pass psi.FileName
                = @"<Replace this with the path to the Central.exe, for example,
C:\Program Files (x86)\Schneider Electric\SoMachine Software\>Central.exe>";
                psi.Arguments = "--noui --enablescripttracing --
AdvancedPythonFunctions --runscript=\
"<Replace this with the full file path where the script is stored,
for example, D:\MyScripts\TestScript.py>"";
                // Create new process and set the starting information Process p =
                new Process();
                p.StartInfo = psi;
                // Set this so that you can tell when the process has completed
                p.EnableRaisingEvents = true;
                p.Start();
                // Wait until the process has completed
                while (!p.HasExited)
                {
                     System.Threading.Thread.Sleep(1000);
                }
```

```
                }
                catch (Exception ex)
                {
                   Console.Write(ex.Message);
                }
                Console.ReadKey();
            }
        }
    }
```

Методы наиболее успешной практики

Наиболее успешная практика

Рассмотрите применяющиеся методы, которые могут помочь Вам избегать ошибок в своем коде программы:

- Функциональные блоки создаются без использования фигурныех скобок ({}), но с добавлением отступа блоков кода.

Примеры:

Язык программирования Си	Язык программирования Python

`int factorial(int x)`
`{`
`if (x > 1)`
`return x* factorial(x - 1); else`
`return 1;`
`}`
	`def factorial(x): if x > 1:`
	`return x * factorial(x - 1) else:`
	`return 1`

- Будьте осторожны при использовании команд копирования и вставки.

  Символы табуляции внутренне заменяются 8 пробелами. Так как многочисленные редакторы по умолчанию используют 4 пробела, это может привести к ошибкам в коде, которые трудно найти. Блоки исходного кода, кажется, имеют тот же размер отступа, но на самом деле добавление отступа отличается.

  Чтобы помочь избежать этого, сконфигурируйте свой редактор таким способом, при котором он автоматически заменял табуляции пробелами.
- Рассмотрите чувствительность к регистру (даже в командной строке).
- Будьте осторожны при вводе путей.
- Не забывайте вводить закрывающие кавычки в конце командной строки (`-- runscript=""`).
- Будьте осторожны с управляющими символами, которые часто используются в путях. Управляющему символу предшествует символ наклонной черты влево (\). Чтобы деактивировать этот эффект, вставьте `r` перед кавычкой.

Пример:

```
project_path = r"D:\PythonProjects\SetParameter.project"
```

- Удостоверьтесь, что объявления цикла и условия заканчиваются двоеточием.

Пример:

```
if len(messages) == 0:
      print("--- Build successful ---")
```

Чтение документации API.NET

Чтение документации API.NET для Python программистов

Текущая предварительная версия документации интерфейса сценария автоматически генеририруется из постилки .NET / C# исходных кодов. Таким образом, она содержит некоторые идиомы, которые не характерны для Python программистов.

Список дает некоторые подсказки, как перевести их в образ мыслей Pythonic:

- Интерфейсы в .NET - контракт, о котором участники (методы, свойства) должны быть обеспечены с помощью классов, реализующих этот интерфейс. В IronPython можно реализовать один или несколько интерфейсов .NET, получив их, как каждый делает с базовыми классами. Когда участник, объявленный интерфейсом, будет отсутствовать в объявлении, исключение будет выдано во времени выполнения. (Пример *DeviceImportFromSvn.py* показывает класс, реализовывая интерфейс `ImportReporter`.)
- В .NET все параметры, свойства и значения, возвращаемые функциями, статически типизированы. Разрешённый тип аннотируется перед именем параметра. Для функций тип возвращаемого значения перед именем функции. Экземпляры подклассов разрешены, когда упомянут родительский класс (или интерфейс). `void` обозначает функцию без возвращаемого значения
- Методы могут быть перегружены, у класса может быть несколько методов с тем же именем, но они отличаются по числу и/или типам параметров. IronPython автоматически вызовет соответствующий вариант.
- Тип INT может содержать целое число между -2 147 483 648... 2,147,483,647, BOOL равен BOOL типа Python (TRUE и FALSE), тип STRING равен `str` типа Python и unicode (которые равны в IronPython). IDictionary<Object, Object> обозначает нормальный словарь Python. IronPython осуществляет автоматическое преобразование между типами .NET и Python.
- Если тип `T` происходит из `IBaseObject <t>`, этот тип может быть расширен с помощью большего количества участников другими плагинами. Фактические использования этого типа `T` в параметрах или возвращаемых значениях будут отмечены в `IExtendedObject<T>`.
- Интерфейс `IEnumerable<T>` описывает любую последовательность (списки, массивы, генераторы), которые приводят только к объектам типа `T` (или подклассам). Когда последовательность приведет к несовместимому объекту, исключение будет выдано во времени выполнения.
- Интерфейс `IList <T>` описывает список, который содержит только объекты типа `T` (или подклассы). При попытке добавить несовместимый объект, будет выдано исключение.
- Синтаксис `params T [] name` равен синтаксису Python `*name` для переменных списков аргументов, но ограничивает параметры типом `T` (или подклассами).

- Перечисления (ENUM) не существуют как конструкция языка в Python. Они используются, чтобы определить установленное количество постоянных значений, например, дни недели. К перечислимым величинам, определенным в .NET, можно получить доступ в IronPython через синтаксис Name.Member (подобный статическим участникам класса), например, `OnlineChangeOption.Try`. Есть несколько образцов эмуляции перечислений в Python, например, *http://pypi.python.org/pypi/enum/* или *http://www.ironpython.info/index.php/Enumerations*.
- Свойства, отмеченные с { get; set; } доступны по чтению-записи, свойства, отмеченные только с { get;} доступны только для чтения. Они аналогичны декоратору @property в Python.

Для сценариев доступны следующие точки входа:

- `system`: Основная функциональность для интеграции в системе SoMachine. Этот объект обеспечивает все функции, описанные под *ISystem Interface*, такие как выход из SoMachine, доступ к окну сообщения или запросу, если `--noUI` mode работает при помощи команды `ui-present`.
- `projects`: Основная функциональность для управления проектами. Этот объект обеспечивает все функции, описанные под *IScriptProjects Interface*, такие как загрузка проектов и архивов проектов. Кроме того, это - точка входа к отдельным проектам.
- `online`: Основная функциональность для онлайнового соединения с устройством. При помощи метода `create_on-line_application` может быть создан онлайновый объект объекта приложения. Этот онлайновый объект позволяет вход в контроллеры, запуск приложений и получение значений переменных.

Точки входа

Детальная информация о точках входа

Драйвер	Имя (Тип)	Описание
System	system (ISystem)	основная функциональность для интеграции в системе SoMachine
	Severity	ENUM для приоритета новостей
	Guid	тип данных для **G**lobally **u**nique **id**entifier (глобально уникального идентификатора)
	PromptResult	ENUM для возвращаемых значений для пользовательского запроса
	MultipleChoiceSelector	делегирует тип для выбора из многочисленного перечня запросов
	PromptChoiceFilter	
Projects	projects (IScriptProjects)	основная функциональность для управления проектами
	ExportReporter	интерфейс для обработки событий во время экспорта
	ImportReporter	интерфейс для обработки событий во время импорта
	ConflictResolve	ENUM для разрешения конфликта во время импорта
Online	online (IScriptOnline)	основная функциональность для онлайнового соединения с устройством
	OnlineChangeOption	ENUM для загрузки типов во время входа в устройство
	ApplicationState	ENUM для состояния приложения
	OperatingState	ENUM для состояния работы
	ValuesFailedException	исключение из-за ошибок обнаруженных с онлайновыми выражениями
	TimeoutException	исключение на тайм-ауте во время онлайновых операций
DeviceObject	DeviceID	инкапсуляция типа для идентификации устройства

Раздел C.2

Примеры механизма выполнения сценариев Schneider Electric

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Параметры устройства	834
Версия компилятора	836
Профиль визуализации	837
Обновление библиотек	838
Очистка и построение приложения	839
Коммуникационные параметры настройки	840
Сообщения диагностики перезапуска	840
Перезагрузка контроллера	841
Convert Device Преобразование устройства	842
Сравнение проектов	845
Усовершенствованные функции управления библиотеками	846
Доступ к POUs	847

Параметры устройства

Обзор

Чтобы изменить параметр, требуются ID параметра и `ParameterSet`.

Чтобы найти требуемое устройство и перечислить соответствующие параметры, используйте метод нахождения, который находит объекты по заданному имени или пути в проекте.

Пример механизма выполнения сценария

```
# We enable the new python 3 print syntax
From___future___import print_function
# The path to the project
project_path = r"D:\PythonProjects\SetParameter.project"
# Clean up any open project:
if projects.primary:
    projects.primary.close()
# Load the project
proj = projects.open(project_path);
# Set the project as primary project proj = projects.primary
# To set a parameter you need a device, a parameter and a new value that should be
assigned to the parameter
# At first search for the SERCOSIII node... sercos Node = proj.find('SERCOSIII',
True)[0]
# ... and add a device named Robot_XK29 to the node, assuming that no other device is
below the SERCOSIII node, otherwise the index must be c hanged
sercosNode.insert("Robot_XK29", 0, DeviceID(4096, "1003
0082", "1.36.2.2"), 'LXM62DxS')
# Now get the children of the SERCOS node, assuming that Robot_XK29 is the only one,
otherwise the index must be changed
Robot_XK29 = sercosNode.get_children(True)[0]
# Call the get_all_parameters() function, to get a complete list of all parameters of
that device object.
# A parameter can contain subparameter, which can be checked with
.HasSubElements property
parameter_list = treeobj.get_all_parameters()
# prints all parameters
```

```
for parameter in parameter_list:
    print("ID: " + parameter.Identifier + " Name: " + parameter.Visible
Name + " Value: " + parameter.Value + " ParameterSet: " +
str((parameter.GetAssociatedConnector).ConnectorId))
# Get the WorkingMode parameter:
# ID: 191 Name: WorkingMode Value: 1 ParameterSet: 1
working_mode = Robot_XK29.get_parameter(191, 1)
# Finally set the WorkingMode parameter to 2 = Deactivated
Robot_XK29.set_parameter(working_mode, "2")
```

Версия компилятора

Обзор

С расширением версии компилятора Вы можете отразить отображённые версии компилятора, и Вы можете установить новую версию компилятора, выполнив сценарий.

Пример механизма выполнения сценария

```
# Enable the new python 3 print syntax
from   future   import print_function
# The path to the project
project_path = r"D:\PythonProjects\GetCompilerVersion.project"
# Clean up any open Project:
ifprojects.primary:
   projects.primary.close()
# Load the project
proj = projects.open(project_path);
# Set the project as primary project
proj = projects.primary
print("All compiler versions")
# Get all compiler versions (filtered)
compiler_versions = compiler_settings.get_all_compiler_versions()
# Print all compiler versions (filtered)
for version in compiler_versions:
    print (" - OEM mapped version: " + version)
# Get active compiler version
compiler_version = compiler_settings.active_compiler_version
print("Current compiler version:" + compiler_version)
# Set new compiler version
compiler_settings.active_compiler_version = "3.5.0.20"
print("New compiler version: " + compiler_settings.active_compil- er_version)
# Save project
projects.primary.save()
```

Профиль визуализации

Обзор

С расширением профиля визуализации Вы можете отобразить профили визуализации, и Вы можете установить активный профиль визуализации проекта.

Пример механизма выполнения сценария

```
# Enable the new python 3 print syntax
from   future   import print_function
#The path to the project
project_path = r"D:\PythonProjects\GetVisualizationProfile.project"
# Clean up any open Project:
if projects.primary:
    projects.primary.close()
# Load the project
proj = projects.open(project_path);
# Set the project as primary project
proj = projects.primary
# Print the active profile
print("Current visual profile: " + visualization_settings.active_pro- file_name)
# Get all available visualization profiles
profile_names = visualization_settings.get_all_visual_profile_names()
# Print the profiles
for visual_profile in profile_names:
    print (" - " + visual_profile)
# Set the profile to V1.35.12.0
visualization_settings.active_profile_name = "V1.35.20.0"
# Get the active profile
profile_name = visualization_settings.active_profile_name
# Print the active profile
print("New visual profile: " + profile_name)
# Save project
projects.primary.save()
```

Обновления библиотек

Обзор

С расширением обновления библиотек Вы можете обновить библиотеки проекта автоматически. Это - та же функция, которая обеспечивается командой **Libraries** → **Automatic version mapping (all libraries)** в графическом интерфейсе пользователя SoMachine.

Пример механизма выполнения сценария

```
# Enable the new python 3 print syntax
from   future   import print_function
# The path to the project
project_path = r"D:\PythonProjects\Example.project"
# Clean up any open project:
if projects.primary:
    projects.primary.close()
# Load the project
proj = projects.open(project_path);
# Set the project as primary project
proj = projects.primary
# Search for die library manager objects in the project
lib_managers = [i for i in proj.get_children(True) if i.is_libman]
# Make the auto mapping for each library manager found for lib_manager in
lib_managers:
    lib_manager.make_auto_mapping()
```

Очистка и построение приложения

Обзор

С расширением очистки и построения приложения, Вы можете очистить проект или разработать новый проект.

Пример механизма выполнения сценария

```
# Enable the new python 3 print syntax
From___future___import print_function
# The path to the project
project_path = r"D:\PythonProjects\Example.project"
# Clean up any open project:
if projects.primary:
    projects.primary.close()
# Load the project
proj = projects.open(project_path);
# Set the project as primary
project proj = projects.primary
# Fetch the active application.
app = proj.active_application
# Clean application new_project.clean_application(app)
# Compile application and store compiler messages in a list messages =
new_project.compile_application(app)
# If messages == None the build was successful
if len(messages) == 0:
   print("--- Build successful ---")
# Otherwise print results
else:
    for i in messages:
        # If serverity == 'Script Engine Exception' the plugin caused an exception.
        # The text describes the exeption details.
        print(i.Serverity, i.Text)
```

Коммуникационные параметры настройки

Обзор

Этот пример показывает, как загрузить или установить IP-адрес контроллера по Вашему выбору, выполнив сценарий.

Пример механизма выполнения сценария

```
from__future import print_function
def main():
    if not projects.primary:
        system.ui.error("No active project.")
        return
    project = projects.primary
    #find the controller by the object name where address should be set and read.
    controller = project.find('LMC', True)[0]
    #reboot the controller controller.set_communication_address('192.168.2.25')
    #read address back and show it.
    print('get_communication_address:=' + controller.get_communica-
    tion_address())
    system.ui.info("Test complete")
main()
```

Перезапуск диагностических сообщений

Обзор

После входа в приложение, Вы можете сбросить диагностические сообщения контроллера. Это - дополнительный метод объекта контроллера.

Следующий пример показывает, как сбросить диагностические сообщения. Вы должны получить основной проект и войти в приложение, как показано в других примерах (создание приложения *(см. стр. 839)*).

Пример механизма выполнения сценария

```
# get the project instance and log in to the application
# find the controller which messages shall be reset
controller = project.find("LMC", True)[0]
# Get all testelements from testseries "TS_Crank"
controller.reset_diagnosis_messages()
```

Перезагрузка контроллера

Обзор

Как только есть экземпляр объекта контроллера в сценарии, Вы можете перезагрузить контроллер при помощи метода на том объекте.

Пример механизма выполнения сценария

```
from__future___import print_function
def perform_application_login(project):
    app = project.active_application
    onlineapp = online.create_online_application(app)
    onlineapp.login(OnlineChangeOption.Try, True)
def main():
    if not projects.primary:
        system.ui.error("No active project.")
        return
    perform_application_login(project)
    #find the controller named 'LMC' which shall be rebooted
    controller = project.find('LMC', True)[0]
    #reboot the controller controller.reboot_plc()
    system.ui.info("Test complete")
main()
```

Преобразование устройства

Обзор

Преобразование устройств в проекте может стать сложной процедурой. Этот API упрощает процесс преобразования и помогает избежать ошибок.

Использование объекта `DeviceID`

API преобразования использует объект `DeviceID`, который идентифицирует устройство или модуль устройства определенной версией. `DeviceID` создается следующим образом:

<device type> <device model> <device version> <module name>

Элемент	Пример	Описание
device type	`4096`	идентифицирует контроллер
device model	`1003 0082` или `1003 009D`	для LMCx00C или LMCx01C, соответственно
device version	`1.50.0.4`	версия фирменного программного обеспечения контроллера
module name	`LXM52`	целевой модуль устройства

API преобразования принимает `DeviceID` как экземпляр объекта или как единственный параметр. Это позволяет использовать `DeviceID`, содержащий все элементы, упомянутые в приведенной выше таблице, или передавать каждый элемент как единственный параметр.

Следующий пример показывает, как создать `DeviceID` для контроллера LMCx00C с версией фирменного программного обеспечения 1.50.0.4:

```
Lmcx00c = DeviceID (4096, "1003 0082", "1.50.0.4")
```

Тестирование, может ли устройство быть преобразовано

Следующий сценарий позволяет Вам проверить, возможно ли преобразование в данную версию, до преобразования устройства.

```
from__future__import print_function
defmain():
    # Set the project as primary project
    proj = projects.primary
    controller = proj.find('LMC', True)[0]
    drive = proj.find('Drive', True)[0]
    # test if controller can be converted using DeviceID
    x01c = DeviceID(4096, "1003 009D", "1.36.2.6")
    if controller.can_convert(x01c):
        system.ui.info("Conversion to LMCx01C possible")
    # test if drive can be converted using Parameters and module
    id if drive.can_convert(4096, "1003 0082", "1.36.2.6", "LXM52"):
        system.ui.info("Conversion to LXM52 possible")
main()
```

Получение альтернативных целей преобразования

API обеспечивает вызов, который получает возможные цели преобразования для определенного устройства. Он возвращает `DeviceID` для каждой цели.

```
from___future___import print_function
#help function to print the delivered device ids def deviceid_to_string(devId):
    mystr = "ID: {0.id} Type: {0.type} Version:
{0.version}".format(devId)
    if hasattr(devId, 'module_id') and devId.module_id is not None:
        mystr += " ModuleID: {0.module_id}".format(devId)
    return mystr
def main():
    if not projects.primary:
        system.ui.error("No active project. Please open the project
        PythonTestProject.projectarchive")
        return
    # Set the project as primary project
    proj = projects.primary
    controller = proj.find('LMC', True)[0]
    alternativeControllers = controller.get_alternative_devices()
    print("ALTERNATIVE DEVICES FOR LMC")
    for id in alternativeControllers:
```

```
        print(deviceid_to_string(id))
    drive = proj.find('drive', True)[0]
    alternativeDrives = drive.get_alternative_devices()
    print("ALTERNATIVE DEVICES FOR DRIVE")
    for id in alternativeDrives:
        print(deviceid_to_string(id))
    system.ui.info("Test complete. Please check the script output
window")
main()
```

Преобразование устройства

Процесс преобразования устройства несложный, потому что единственное требуемое действие - вызыв метода преобразования.

```
from___future__import print_function
def main():
    proj = projects.primary
    controller = proj.find('LMC', True)[0]
    drive = proj.find('Drive', True)[0]
    # converting the controller
    controller.convert(4096, "1003 009D", "1.36.2.6")
    # converting the drive
    drive.convert(DeviceID(4096, "1003 0082", "1.50.0.4"), "LXM52")
main()
```

Сравнение проектов

Обзор

Есть несколько вариантов использования, где полезно иметь сценарий, автоматически сравнивающий содержание 2 проектов. Функция сравнения проекта Python позволяет Вам сравнивать 2 проекта. В результате она предоставляет информацию, если проекты отличаются, а также подробное дерево XML, которое отражает дерево проекта и показывает различия для каждого объекта.

Пример механизма выполнения сценария

```
from__future__ import print_function
def main():
      proj = projects.primary
      # compare the Primary Project to another Project on disk
      diff = proj.compare_to("CompTest_Right.project")
      write_diff(diff, "Diff1.xml")
      # compare, but ignore whitespaces, comments and properties
      diff = proj.compare_to("CompTest_Right.project", True, True, True)
      write_diff(diff, "Diff2.xml")
def write_diff(differences, filename):
      if differences.DifferenceFound:
          f = open(filename, 'wb')
          f.writelines(differences.ResultTree)
main()
```

Усовершенствованные функции управления библиотеками

Обзор

Logic Builder SoMachine предлагает усовершенствованные функции для управления библиотеками, так называемых прямо совместимых библиотек *(см. SoMachine, Functions and Libraries User Guide)*. Они обеспечивают удобный способ, чтобы управлять ссылками и зависимостями среди библиотек.

Эта функциональность также доступна с помощью сценариев и может использоваться в **Library Manager** (менеджере библиотек) всего проекта или в отдельном приложении в проекте. Следующий сценарий показывает, как проверить библиотеки на прямую совместимость и допустимые ссылки. Он автоматически отображает ссылки и явно устанавливает версии библиотеки.

Пример механизма выполнения сценария

```
p = projects.primary
app = p.active_application
libmgr = app.get_library_manager()
print("# Checking all libraries:")
for lib in libman.get_libraries():
    print("-    " + lib + " Is Forward Compatible Library?     "
+ str(libmgr.is_library_forward_compatible(lib)))
if not libmgr.is_current_mapping_valid():
    for lib in libmgr.get_invalid_library_mappings():
        print("Library reference cannot be satisfied for: " + lib)
    print("Trying to auto-map libraries to valid versions")
    libmgr.make_auto_mapping()
else:
    print("All mappings valid")
# set version using individual parameters
libmgr.set_new_library_version("PD_GlobalDiagnostics",
"Schneider Electric", "1.0.1.0")
# set version using the library full name
libmgr.set_new_library_version("PD_AxisModule, 1.1.6.0 (Schneider
Electric)", "1.2.4.0")
# set version to Legacy
libmgr.set_new_library_version("PD_Template", "Schneider Electric",
None)
```

Доступ к POUs

Обзор

Следующий пример показывает, как распечатать и управлять кодом POU. Это доступно только для текстовых языков программирования.

Пример механизма выполнения сценария

```
if not projects.primary:
    system.ui.error('No primary project set')
p = projects.primary
pou = p.find('SR_Main', True)[0]
# read and print the declaration of the program
decl = pou.get_interface_text()
print(decl)
# read and print the implementation of the program
code = pou.get_implementation_text()
print(code)
decl = "PROGRAM SR_Main\n" + \
            "VAR\n" + \
            "   iTest: INT;\n" + \
            "END_VAR";
code = "iTest := iTest +1;"
# write new code to the declaration and implementation
pou.set_interface_text(decl)
pou.set_implementation_text(code)
```

Раздел C.3

Примеры механизма выполнения сценария CoDeSys

Обзор

Эта глава обеспечивает примеры часто используемых членов процессора CoDeSysScript (CoDeSysScript Engine). Для полного описания членов каждого пространства имен обратитесь к описанию CoDeSys API.

Содержание этого раздела

Этот раздел содержит следующие темы:

Тема	Страница
Проект	849
Онлайн-приложение	854
Объекты	857
Устройства	858
Система / Интерфейс пользователя (UI - User Interface)	860
Чтение значений	862
Чтение значений из рецепта и посылка электронного письма	863
Определение дерева устройства открытого проекта	865
Пример сценария 4: Импорт устройства в PLCOpenXML из Subversion	866

Проект

Обзор

Так как примеры для этого пространства имен относительно коротки и очевидны, их значение не объяснено подробно. Полные примеры обеспечены, где это необходимо.

Новый проект

Этот метод создает новый проект. Он состоит из 2 параметров:

- последовательность, определяющая расположение, где проект будет сохранен
- логический параметр: если TRUE, проект будет новым основным проектом. Этот параметр дополнительный, значение по умолчанию - TRUE.

Метод возвращается экземпляр `IProject` *(обратитесь к спецификации в документе Automation Platform SDK)*, который может использоваться для дальнейших шагов.

```
# Creates a new project
proj = projects.create("C:\PythonProjects\Example.project", True)
```

Загрузка проекта

Этот метод загружает проект. Любые открытые проекты не будут закрыты.

Первый параметр определяет путь проекта, который будет загружен.

```
# Load the project
proj = projects.open(project_path)
```

Сохранение проекта

Этот метод сохраняет проект в своем физическом расположении.

```
# Save project
projects.primary.save()
```

Сохранение архива

Этот метод сохраняет проект как архив. Дополнительные категории, которые выбраны по умолчанию, включены, но никаких дополнительных файлов нет.

Первый параметр определяет путь, где архив будет сохранен.

```
# Save archive
projects.primary.save_archive("D:\Archive\Example.archive")
```

Закрытие проекта

Этот метод закрывает проект. Если есть несохранённые изменения в этом проекте, то эти изменения будут отменены.

```
# Clean up any open project:
if projects.primary:
    projects.primary.close()
```

Поиск объектов

Этот метод находит объекты, соответствующие заданному имени. Он состоит из 2 параметров:

- Первый параметр - имя объекта, который ищется.
- Второй параметр определяет, выполнен ли рекурсивный поиск. Этот параметр дополнительный, значение по умолчанию - FALSE. Этот метод возвращает набор объектов.

```
device = proj.find('DRV_Lexium62', True)[0]
```

Имена в дереве не уникальны. Это имеет эффект, с помощью которого могут быть найдены несколько объектов. Поиск против нелокализованного имени.

Собственный импорт

Этот метод импортирует указанные файлы в собственном формате XML на верхний уровень этого проекта.

```
system.trace
import os
project_path = r"D:\MyProjects\Example.project"
import_path = r"D:\MyProjects\ImportFiles"
# Close project if opened
if projects.primary:
    projects.primary.close()
proj = projects.open(project_path);
# Set the new project to primary proj=projects.primary
files = os.listdir(import_path)
# create the import reporter
class Handler(NativeImportHandler):
    def conflict(self, name, obj, guid):
        print("Object already exists: ",name)
        return NativeImportResolve.skip
    def progress(self, name, obj, exception):
        print("in progess: ", name)
    def skipped(self, list):
        for obj in list:
            print("Skipped: ", obj.get_name())
```

```
def importFilter(name, guid,type,path):
    return True;
# create the importer instance.
handler = Handler()
for file in files:
    file_path = import_path + "\\" + file
    print(file)
    proj.import_native(file_path, importFilter, handler)
proj.save();
```

Импорт PLCOpenXML

Этот метод импортирует содержание указанного файла PLCopenXML на верхний уровень проекта.

```
# CoDeSys XML import/export functionality.
from__future__import print_function
import sys, io
# Set target Project to primary proj=projects.primary
# Create the import reporter
class Reporter(ImportReporter):
    def error(self, message):
        system.write_message(Severity.Error, message)
    def warning(self, message):
        system.write_message(Severity.Warning, message)
    def resolve_conflict(self, obj):
        return ConflictResolve.Copy
    def added(self, obj):
        print("added: ", obj)
    def replaced(self, obj):
        print("replaced: ", obj)
    def skipped(self, obj):
        print("skipped: ", obj)
    @property
    def aborting(self):
        return False
# Create the importer instance.
reporter = Reporter()
filename = r"D:\ExportObjects\Drv_Master.xml"
# Search for the SERCOSIII node, where the device should be added.
device = proj.find('SERCOSIII', True)[0]
# Import the data into the project.
device.import_xml(reporter, filename)
```

Собственный экспорт

Этот метод экспортирует данные объекты в собственном формате в последовательность или файл по данному пути. Неэкспортные объекты обнаруживаются как ошибки, но экспорт продолжается.

```
# Tests CoDeSys native import/export functionality.
From__ future__ import print_function
project_path = r"D:\MyProjects\Example.project"
# Clean up any open Project:
if projects.primary:
    projects.primary.close()
proj = projects.open(project_path); proj=projects.primary
import sys, io
# Collect all POU nodes in that list.
projectObjects = []
# Collect all the leaf nodes.
for node in proj.get_children(True):
    projectObjects.append(node)
# Print everything just to know what is going on.
for i in projectObjects:
    print("Found: ", i.type, i.guid, i.get_name())
# Export the files.
for candidate in projectObjects:
    # Create a list of objects to export:
    # The object itself
    objects = [candidate]
    # And sub-objects (POUs can have actions, properties, ...)
    objects.extend(candidate.get_children(True))
    # And the parent folders.
    parent = candidate.parent
    while ((not parent.is_root) and parent.is_folder):
        objects.append(parent)
        parent = parent.parent
    # Create a unique file name:
    filename = "D:\ExportFiles\\%s %s.export" % (candidate.get_name(),
    candidate.guid)
    # print some user information
    print("Exporting ", len(objects), " objects to: ", filename)
    # and actually export the project.
    proj.export_native(objects, filename)
```

Экспорт PLCOpenXML

Этот метод экспортирует заданные объекты в формате PLCopenXML в последовательность или файл по заданному пути. Неэкспортные объекты обнаруживаются как ошибки, но экспорт продолжается.

```
# CoDeSys XML import/export functionality.
from__ future__ import print_function
import sys, io
# Set target Project to primary proj=projects.primary
# define the printing function
def printtree(treeobj, depth=0):
    name = treeobj.get_name(False)
    if treeobj.is_device:
       deviceid = treeobj.get_device_identification()
       print("{0} - {1} {2}".format("  "*depth, name, deviceid))
    for child in treeobj.get_children(False):
       printtree(child, depth+1)
for obj in projects.primary.get_children():
    printtree(obj)
# Create the export reporter
class Reporter(ExportReporter):
    def error(self, message):
        system.write_message(Severity.Error, message)
    def warning(self, message):
        system.write_message(Severity.Warning, message)
    def nonexportable(self, message):
        print(message)
    @property
        def aborting(self):
    return False
reporter = Reporter()
# Finds the DRV_Master object in the project
device = proj.find('DRV_Master', True)
filename = r"D:\ExportObjects\Drv_Master.xml"
# Exports the object to the hard drive
proj.export_xml(reporter, device, filename, True, True)
```

Онлайн-приложение

Обзор

ПРИМЕЧАНИЕ: Некоторые команды онлайн-приложения могут временно изменить активное приложение. Этот интерфейс экспортируется в Python, и таким образом соответствует стандартам наименования Python.

Создание онлайн-приложения

Этот метод создает онлайн-приложение.

```
app = online.create_online_application();
```

Подготовленные значения

Этот пример показывает, как записать подготовленные значения.

```
# Tests for the Online functionality - listing of prepared values.
import utils
try:
    print "Trying to create online application."
    app = online.create_online_application();
    except Exception as e:
    print "Currently offline, executing startup..."
    execfile(utils.makepath("OnlineTestStartup.py"))
print "Retrying to create online application."
app = online.create_online_application();
print "app:", app.application_state, "op:",
app.operation_state.ToString("f")
print "Unpreparing values:"
for expression in app.get_prepared_expressions():
    app.set_prepared_value(expression, '')
    print "%s: '%s' '%s'" % (expression,
app.read_value(expression),
app.get_prepared_value(expression))
print "Unforcing values:"
for expression in app.get_forced_expressions():
    app.set_unforce_value(expression)
    print "%s: '%s' '%s'" % (expression, app.read_value(expression),
    app.get_prepared_value(expression))
app.force_prepared_values()
assert len(app.get_prepared_expressions()) == 0, "still some prepared
values remain..."
```

```
assert len(app.get_forced_expressions()) == 0, "still some prepared
values remain..."
print "now preparing a value and forcing it:"
app.set_prepared_value("POU.testoutput", "4711");
app.force_prepared_values()
print "now preparing a value and writing it:"
app.set_prepared_value("POU.testint", "INT#1147");
app.write_prepared_values();
print "The prepared values are now written."
```

Выполнение входа в приложение

Этот метод выполняет вход в приложение. Если в приложение вошли прежде, из него будет осуществлён выход, и будет выполнен новый вход.

Он состоит из 2 параметров:

- Первый параметр - опция изменения.
- второй параметр удалит предыдущие приложения, если установлен в `True`.

```
project_path = r"D:\MyProjects\Example.project"
# Clean up any open Project:
if projects.primary:
    projects.primary.close()
proj = projects.open(PROJECT); proj = projects.primary
# Fetch the active application.
app = proj.active_application
# Create the online application for it.
onlineapp = online.create_online_application(app)
# Log in to the device.
onlineapp.login(OnlineChangeOption.Try, True)
```

Выход из приложения

Этот метод выводит из приложения. Если приложение не авторизовано, ничего не происходит.

```
# Logout.onlineapp.logout()
```

Запуск приложения

Этот метод запускает приложение.

```
project_path = r"D:\MyProjects\Example.project"
# Clean up any open Project:
if projects.primary:
    projects.primary.close()
proj = projects.open(PROJECT);
proj = projects.primary
# Fetch the active application.
app = proj.active_application
# Create the online application for it.
onlineapp = online.create_online_application(app)
# Log in to the device.
onlineapp.login(OnlineChangeOption.Try, True)
# Start the application, if necessary.
if not onlineapp.application_state == ApplicationState.run:
    onlineapp.start()
# Let the app do its work for some time...
system.delay(1000)
```

Остановка приложения

Этот метод останавливает приложение.

```
# Stop the application
onlineapp.stop()
```

Объекты

Find- поиск

Этот метод находит объекты, соответствующие заданному имени. Он состоит из 2 параметров:

- Первый параметр - имя объекта, который ищется
- Второй параметр определяет, выполняется ли рекурсивный поиск. Этот параметр дополнительный, значение по умолчанию - FALSE. Этот метод возвращает набор объектов.

```
# Finds the DRV_Master object in the project
device = proj.find('DRV_Master', True)
```

Remove - удаление

Этот метод удаляет объект

```
# Finds the DRV_Master object in the project
device = proj.find('DRV_Master', True)
# Removes the DRV_Master object from the project
device.remove()
```

Rename - переименование

Этот метод переименовывает объект новым именем.

```
# Finds the DRV_Master object in the project
device = proj.find('DRV_Master', True)
# Removes the DRV_Master object from the project
device.rename('DRV_Master_2')
```

Импорт/Экспорт

Обратитесь к описанию импорта/экспорта для проектов *(см. стр. 851)*. Единственная разница - то, что объект вызывают для импорта / экспорта вместо проекта.

Устройства

Обзор

Этот раздел описывает методы для управления объектами устройства.

Add - добавление

Этот метод добавляет указанное устройство. Он состоит из 3 параметров:

- строки, определяющей имя устройства
- `DeviceID`, определяющего ID устройства
- строки, определяющей ID модуля

```
device.insert("RobotA", DeviceID(4096, "1003 0082", "1.36.1.1"),
'LXM62DxS')
```

Disable - отключение

Этот метод отмечает это устройство, как отключенное во время загрузки.

```
# Finds the DRV_Master object in the project
device = proj.find('DRV_Master', True)[0]
device.disable()
```

Enable - включение

Этот метод отмечает это устройство, как включенное во время загрузки.

```
# Finds the DRV_Master object in the project
device = proj.find('DRV_Master', True)[0]
device.enable()
```

Get Address - получение адреса

Этот метод получает адрес устройства. Он возвращает строку.

Finds the DRV_Master object in the project

device = proj.find('DRV_Master', True)[0] device.get_address()

Get Device Identification - получение идентификации устройства

Этот метод получает идентификацию устройства.

```
# Finds the DRV_Master object in the project
device = proj.find('DRV_Master', True)[0]
deviceid = device.get_device_identification()
print("{0} - {1} {2}".format("          "*depth, name, deviceid))
```

Get Gateway - получение шлюза

Этот метод возвращает GUID шлюза.

```
# Finds the DRV_Master object in the project
device = proj.find('DRV_Master', True)[0]
gateway = device.get_gateway()
```

Insert - вставка

Этот метод вставляет указанное устройство по указанному индексу. Он состоит из 4 параметров:

- строка, определяющая имя устройства
- `Int32`, определяющий индекс, где вставить устройство
- `DeviceID`, определяющий ID устройства
- строка, определяющая ID модуля

```
device.insert("RobotA", 1, DeviceID(4096, "1003 0082", "1.36.1.1"),
'LXM62DxS')
```

Set Gateway and Address - Установка шлюза и адреса

Этот метод устанавливает шлюз и адрес. Если Вы передадите пустой GUID и пустой адрес, адрес шлюза будет очищен.

device.set_gateway_and_address(GUID gateway, string address)

Set Simulation Mode - установка режима симулятора

Этот метод устанавливает режим симулирования. Если параметр установлен в True, симулирование включено.

```
# Finds the DRV_Master object in the project
device = proj.find('DRV_Master', True)[0] device.set_simulation_mode (True)
```

Update - обновление

Этот метод обновляет указанное устройство.

```
# Finds the DRV_Master object in the project
device = proj.find('DRV_Master', True)[0]
device.update(DeviceID(4096, "1003 0082", "1.36.1.1"), 'LXM62DxS')
```

Система / Интерфейс пользователя (UI - User Interface)

Диалоговое окно обзора каталога

Открывает диалоговое окно для просмотра каталога. В режиме `--noUI` Вы можете здесь просто ввести путь.

```
system.ui.browse_directory_dialog("Browse Directory Dialog",
r"D:\Python", Environment.SpecialFolder.Desktop, True)
```

Он содержит 4 параметра:

- строка, содержащая сообщение
- строка, содержащая путь, который будет предварительно выбран, когда откроется диалоговое окно
- `Environment.SpecialFolder` содержит корневую папку для диалогового окна обзора
- логический параметр: Если он `True`, кнопка, разрешающая Вам создать новые папки, отображена в диалоговом окне.

Этот метод возвращает выбранный путь. Если Вы отменяете диалоговое окно, ничто не возвращается.

Choose - выбор

Этот метод позволяет Вам выбирать один из нескольких перечисленных элементов.

```
list = ["RobotA", "RobotB", "RobotC"]
system.ui.choose("Choose", list, True)
```

Он содержит 3 параметра

- строка, содержащая сообщение
- список опций, которые будут выведены на экран: объекты преобразованы в строки, чтобы отображать их.
- логический параметр: Если он `True`, кнопка, разрешающая Вам создать новые папки, отображается в диалоговом окне.

Этот метод возвращает кортеж Python, содержащий 2 элемента:

- индекс выбранного пункта, или
  - 1, если отменяемый был установлен в True, и Вы отменяли диалоговое окно
- выбранный пункт или `None`

Обнаружение ошибки

Этот метод показывает сообщение об обнаружении ошибок. Он запрещает дальнейшие действия, пока сообщение не было подтверждено.

```
system.ui.error("Error")
```

Информация

Этот метод показывает информационное сообщение. Он запрещает дальнейшие действия, пока сообщение не было подтверждено.

```
system.ui.info("Info")
```

Диалоговое окно Open File

Этот метод отображает диалоговое окно **Open File**. В режиме `--noUI` Вы можете здесь просто ввести путь.

```
system.ui.open_file_dialog("Select a file")
```

Query String - запрос строки

Этот метод запрашивает ввод или редактирование текстовой строки.

```
system.ui.query_string("Please enter a string")
```

Он возвращает строку с введенным текстом.

Диалоговое окно сохранения файла

Этот метод выводит на экран диалоговое окно **Save File**. В режиме --noUI Вы можете здесь просто ввести путь.

```
system.ui.save_file_dialog("Python Script: Save File")
```

Предупреждение

Этот метод показывает предупреждающее сообщение. Он запрещает дальнейшие действия, пока сообщение не было подтверждено.

```
system.ui.warning("Warning")
```

Чтение значений

Обзор

Вы можете запустить следующий пример из пользовательского интерфейса SoMachine или из командной строки.

Чтобы запустить его из командной строки, измените путь установки SoMachine на подкаталог **Common** (*<Replace this with the path to the Central.exe*, например, *C:\Program Files (x86)\Schneider Electric\SoMachine Software\>*) и введите команду

```
start /wait Central.exe --
runscript="<Replace this with the full file path
where the script is stored, for example, D:\MyScripts\ReadVariable.py>"
```

Сценарий открывает приложение в SoMachine и входит в устройство. Если контроллер не в режиме RUN, он будет установлен в RUN. Тогда переменная `iVar1` считывается и выводится на экран в виде **Messages** или в командной строке. В конце приложение закрывается.

Пример сценария *ReadVariable.py*

```
# Close all projects
while len(projects.all) > 0: projects.all[0].close()
    # opens project
proj = projects.open("D:\\data\\projects\\Ampel.project")
# set "Ampel.project" to active application
app = proj.active_application
onlineapp = online.create_online_application(app)
# login to device
onlineapp.login(OnlineChangeOption.Try, True)
# set status of application to "run", if not in "run"
if not onlineapp.application_state == ApplicationState.run:
    onlineapp.start()
# wait 1 second system.delay(1000)
# read value of iVar1
value = onlineapp.read_value("PLC_PRG.iVar1")
# display value in message view or command line
print value
# log out from device and close "Ampel.project" onlineapp.logout()
proj.close()
```

Чтение значений из рецепта и посылка электронного письма

Обзор

Вы можете запустить следующий пример из пользовательского интерфейса SoMachine или из командной строки.

Чтобы запустить его из командной строки, измените установки пути SoMachine на подкаталог **Common** (*<Replace this with the path to the Central.exe, for example, C:\Program Files (x86)\Schneider Electric\SoMachine Software\>)* и введите команду.

```
start /wait Central.exe --
runscript="<Replace this with the full file path
where the script is stored, for example, D:\MyScripts\ScriptEmail.py>".
```

Сценарий открывает приложение в SoMachine и входит в устройство. Если контроллер не в режиме RUN, он будет установлен в RUN. Тогда переменная iVar1 считана и выведена на экран в виде **Messages** или в командной строке. В конце приложение закрывается.

Пример сценария ScriptEmail.py

```
# Close current project if necessary and open "ScriptTest.project"
if not projects.primary == None:
    projects.primary.close()
project = projects.open("D:\\Data\\projects\\scriptTest.project")
# retrieve active application
application = project.active_application
# create online application
online_application = online.create_online_application(application)
# login to application.
online_application.login(OnlineChangeOption.Try, True)
# start PLC if necessary
if not online_application.application_state == ApplicationState.run:
    online_application.start()
# wait 2 seconds
system.delay(2000)
# open recipe file to read values.
recipe_input_file = open("D:\\Data\\projects\\RecipeInput.txt", "r")
watch_expressions = []
for watch_expression in recipe_input_file:
    watch_expressions.append(watch_expression.strip())
print watch_expressions
# read values from the controllerd
watch_values = online_application.read_values(watch_expressions)
print watch_values
# open output file to write values
recipe_output_file = open("D:\\Data\\projects\\RecipeOutput.txt", "w") for i in
range(len(watch_expressions)):
    recipe_output_file.write(watch_expressions[i])
```

```
    recipe_output_file.write(" = ")
    recipe_output_file.write(watch_values[i])
    recipe_output_file.write("\n")
# Close files
recipe_input_file.close() recipe_output_file.close()
# send Email
# import respective libraries
import smtplib
from email.mime.text import MIMEText
#open output file
recipe_output_file = open("D:\\Data\\projects\\RecipeOutput.txt", "r")
mail = MIMEText(recipe_output_file.read())
recipe_output_file.close()
#email address sender and recipient
fromm = "info@3s-software.com"
to = "info@3s-software.com"
# set sender and recipient
mail["Subject"] = "Attention value has changed"
mail["From"] = fromm
mail["To"] = to
# send email
smtp = smtplib.SMTP("name of smtp server")
smtp.sendmail(fromm, [to], mail.as_string())
smtp.quit()
# logout and close application
online_application.logout()
project.close()
```

Определение дерева устройства открытого проекта

Обзор

Этот пример определяет объекты в **Devices tree** открытого проекта и распечатывает их в командной строке или виде **Messages**. Он может быть запущен из пользовательского интерфейса SoMachine или из командной строки.
Чтобы запустить его из командной строки, измените установки пути SoMachine на подкаталог **Common** (*<Replace this with the path to the Central.exe, for example, C:\Program Files (x86)\Schneider Electric\SoMachine Software\>)*, и введите команду.

```
start /wait Central.exe --
runscript="<Replace this with the full file path
where the script is stored, for example, D:\MyScripts\DevicePrintTree.py>".
```

Пример сценария *DevicePrintTree.py*

```
# We enable the new python 3 print syntax
from__future__ import print_function
import sys
# define the printing function
def printtree(treeobj, depth=0):
    if treeobj.is_root:
        name = treeobj.path
        deviceid = ""
    else:
        name = treeobj.get_name(False)
        if treeobj.is_device:
            deviceid.get_device_identification()
        else:
            deviceid = ""
    print("{0} - {1} {2}".format("     "*depth, name, deviceid))
    for child in treeobj.get_children(False):
        printtree(child, depth+1)
# Now see whether a primary project is open.
if not projects.primary:
    print("Error: Please open a project file first!", file=sys.stderr)
    sys.exit()
# And the actual output
print("--- The current tree of: ---")
printtree(projects.primary)
print("--- Script finished. ---")
```

Пример сценария 4: Импорт устройства в PLCOpenXML из Subversion

Обзор

Этот пример импортирует устройство в PLCOpenXML от Subversion через командную строку `svn client`. Он может быть запущен из пользовательского интерфейса SoMachine или из командной строки.

Чтобы запустить его из командной строки, измените на подкаталог **Common** пути установки SoMachine (*<Replace this with the path to the Central.exe, for example, C:\Program Files (x86)\Schneider Electric\SoMachine Software\>*) и введите команду
`start wait Central.exe runscript="<Replace this with the full file path where the script is stored, for example, D:\MyScripts\DeviceImportFromSvn.py>"`.

Пример сценария *DeviceImportFromSvn.py*

```
# Imports a Device in PLCOpenXML from Subversion via command line svn client.
# We enable the new python 3 print syntax
from__future__ import print_function
import sys, os
# some variable definitions:
SVNEXE = r"C:\Program Files\Subversion\bin\svn.exe" XMLURL =
"file:///D:/testrepo/testfolder/TestExport.xml" PROJECT = r"D:\test.project"
# clean up any open project:
if projects.primary:
    projects.primary.close()
# Fetch the plcopenxml data from subversion.
# The 'with' construct automatically closes the open pipe for us.
with os.popen('"' + SVNEXE + '" cat ' + XMLURL, 'r') as pipe:
    xmldata = pipe.read()
# create a new project:
proj = projects.create(PROJECT)
# create the import reporter
class Reporter(ImportReporter):
    def error(self, message):
        system.write_message(Severity.Error, message)
    def warning(self, message):
        system.write_message(Severity.Warning, message)
    def resolve_conflict(self, obj):
        return ConflictResolve.Copy
    def added(self, obj):
        print("added: ", obj)
    def replaced(self, obj):
        print("replaced: ", obj)
```

```
def skipped(self, obj):
    print("skipped: ", obj)

@property
def aborting(self):
    return False
# create the importer instance.
reporter = Reporter()
# import the data into the project.
proj.import_xml(reporter, xmldata)
# and finally save. :-)
proj.save()
print("--- Script finished. ---")
```

Приложение D

Управление пользователями для Soft PLC

Обзор

В этом разделе описываются виды **Users and Groups** (пользователи и группы) и **Access Rights** (права доступа) редактора устройств. Эти виды доступны, только если проект SoMachine содержит контроллеры **Soft PLC**.

Содержание этой главы

Эта глава содержит следующие теиы:

Тема	Страница
Общая информация об управлении пользователями для Soft PLC	870
Пользователи и группы	872
Права доступа	875

Общая информация об управлении пользователями для Soft PLC

Обзор

Функция управления пользователями, описанная в этом разделе, позволяет Вам определять учетные записи пользователей и конфигурировать права доступа (полномочия) для контроллеров **Soft PLC**.

Права получить доступ к объектам проекта через указанные действия присвоены только группам пользователей, не единственной учетной записи пользователя. Таким образом, каждый пользователь должен быть членом группы.

Управление пользователями для Soft PLC

Перед установкой пользователей и групп пользователей для контроллеров **Soft PLC**, рассмотрите следующее:

- По умолчанию существует группа **Everyone**. Каждый определенный пользователь или другие группы - автоматически член этой группы. Таким образом, каждой учетной записи пользователя, по крайней мере, автоматически предоставляют настройки по умолчанию. Группа **Everyone** не может быть удалена, только переименована. Участники не могут быть удалены из этой группы. По умолчанию у **Everyone** нет разрешения изменять текущих пользователей, группы и конфигурацию разрешений.
- По умолчанию **Owner** (владелец) группы также существует, содержа одного пользователя **Owner**. В новом проекте первоначально только у **Owner** есть разрешение изменять текущих пользователей, группы и конфигурацию разрешений. Таким образом только **Owner** может присвоить это право другой группе. Первоначально, **Owner** может войти в систему с именем **Owner** и пустым паролем. Пользователи могут быть добавлены к или удалены из группы **Owner**, но по крайней мере 1 пользователь должен остаться. Эта группа – так же как **Everyone** - не может быть удалена. Ей предоставлены все права доступа. Таким образом, не возможно сделать проект неиспользуемым, отклоняя соответствующие права для всех групп. Вы можете переименовать и группу и пользователя **Owner**.
- При запуске системы программирования или запуске проекта, первоначально никакой пользователь не авторизовал (log on) проект. Но затем пользователь может дополнительно войти в систему через определенную учетную запись пользователя с именем пользователя и паролем, чтобы иметь специальный набор прав доступа.
- Полагайте, что у каждого проекта есть свое собственное управление пользователями. Так, например чтобы получить специальный набор прав доступа для библиотеки, включенной в проект, пользователь должен отдельно войти в систему этой библиотеки. Также пользователи и группы, установленные в различных проектах, не идентичны, даже если у них есть идентичные имена.

ПРИМЕЧАНИЕ: Только пользователю **Owner** группы **Owner** разрешено изменять в настоящее время сконфигурированные полномочия, группы и пользователей. Таким образом, только **Owner** может присвоить это разрешение другой группе.

ПРИМЕЧАНИЕ: пользовательские пароли сохраняются безвозвратно. Если Вы не помните пароля, соответствующая учетная запись пользователя становится неприменимой. Если Вы не помните пароля группы **Owner**, весь проект может стать неприменимым.

Управление правом доступа для Soft PLC

Управление пользователями в проекте полезно только в сочетании с управлением правом доступа (=permissions).

Рассмотрите следующее:

- В новом проекте в основном все права еще не определены явно, но установлены в значение по умолчанию. Это значение по умолчанию обычно предоставляют за исключением права изменить текущих пользователей, группы и конфигурацию разрешений. По умолчанию его предоставляют только для группы **Owner**.
- Когда проект создается, член группы с правом изменить полномочия может определить права. Каждое определенное право можно предоставить, или отклонить, или вернуть на значение по умолчанию.
  - Выполните управление правом доступа проектом в **Project** → **User Management** →**Permissions...** → диалоговом окне **Permissions**
  - Выполните управление правом доступа для объектов в **View** → **Properties...** → **Properties -** диалоговом окне **Project Information**, выбирая вкладку **Access control** (управление доступом).
- Права доступа на объектах наследуемы. Если у объекта есть родительский объект (например, если действие присвоено объекту программы, который вставлен в дерево структуры ниже программы, то программа - родительский элемент объекта действия), текущие права родительского элемента автоматически станут настройками по умолчанию дочернего элемента. Родительско-дочерние отношения объектов относительно прав доступа обычно соответствуют отношениям, показанным в **Devices tree**, **Applications tree** и **Tools tree**, и обозначены в диалоговом окне **Permissions** синтаксисом <father object>.<child object>. Пример: Действие `ACT` присвоено объекту POU `PLC_PRG`. Таким образом, в **Applications tree** `ACT` показан в древовидной структуре, расположенным с отступом ниже `PLC_PRG`. В диалоговом окне **Permissions** `ACT` представлен `PLC_PRG.ACT`, указывающим, что `PLC_PRG` - родительский элемент `ACT`. Если право изменения будет явно отклонено для `PLC_PRG` и определенной группы пользователей, значение по умолчанию права изменения для `ACT` будет также автоматически отклонено.

Пользователи и группы

Обзор

Вид **Users и Groups** редактора устройства обеспечено для устройств, поддерживающих онлайновое управление пользователями. Это позволяет устанавливать учетные записи пользователей и группы пользователей, который, в сочетании с управлением правом доступа, служит тому, чтобы управлять доступом на объектах контроллера в режиме онлайн.

Если желательно, чтобы определенные функции контроллера могли быть выполнены только авторизованными пользователями, используйте онлайновую функцию управления пользователями. Это позволяет Вам устанавливать учетные записи пользователей, чтобы присвоить права доступа для групп пользователей и вызвать пользовательскую идентификацию при входе в систему.

Специфичное для устройства управление пользователями может быть предопределено описанием устройства. Это описание также определяет, к какой степени определения могут быть отредактированы в диалоговых окнах конфигурации

В управлении пользователями проекта пользователи должны быть членами групп. Только группы пользователей могут получить определенные права доступа *(см. стр. 875)*.

Использование диалогового окна конфигурации

В основном обработка диалоговых окон управления пользователями подобна этому из управления пользователями проекта. Есть даже возможность импортировать определения учетной записи пользователя из управления пользователями проекта.

Вид **Users and Groups** редактора устройства

Это вид разделён на 2 части:

- Верхняя часть выделена для управления доступом **Users** (пользователей).
- Нижняя часть выделена для управления доступом **Groups** (групп).

Область Users

Следующие кнопки доступны для установки прав доступа пользователя:

Кнопка	Описание
Add	Открывает диалоговое окно **Add User**. Введите имя (**Name)** пользователя и Пароль (**Password)**. Повторите пароль в поле **Confirm password**.
Import	Открывает диалоговое окно **Import Users**. Оно показывает все имена пользователей, которые определены в управлении пользователями проекта. Выберите одну или несколько записей и нажмите **OK**, чтобы подтвердить. Открывается диалоговое окно **Enter password**. Введите соответствующий пароль, поскольку он определен в управлении пользователями проекта. Вы можете тогда импортировать учетную запись пользователя к специфичному для устройства управлению пользователями. Однако эти пароли не импортируются. **ПРИМЕЧАНИЕ**: У каждой импортированной учетной записи пользователя будет пустое определение пароля.
Edit	Изменяет выбранную учетную запись пользователя относительно имени пользователя и пароля. Диалоговое окно **Edit User <user name>** соответствует диалоговому окну **Add User** (см. выше).
Delete	Удаляет выбранную учетную запись пользователя.

Область Groups

Следующие кнопки доступны для установки групп пользователей:

Кнопка	Описание
Add	Открывает диалоговое окно **Add Group**. Введите название группы(**Name)** и выберете из определенных пользователей тех, кто должен быть членами этой группы.
Import	Открывает диалоговое окно **Import Groups**. Оно показывает группы, которые определены в управлении пользователями проекта. Выберите одну или несколько записей и нажмите **OK**, чтобы интегрировать их в списке группы специфичного для устройства управления пользователями.
Edit	Изменяет выбранную группу относительно названия группы и связанных пользователей. Диалоговое окно **Edit Group <group name>** соответствует диалоговому окну **Add Group** (см. выше).
Delete	Удаляет выбранную в настоящее время группу.

Применение и сохранение текущей конфигурации

Кнопки доступны на главной панели диалогового окна:

Кнопка	Описание
Download to device	Загружает текущую конфигурацию управления пользователями в устройство. Она вступит в силу только после того, как она будет загружена.

Кнопка	Описание
Upload from device	Выгружает конфигурацию, применённую в настоящее время на устройстве, в диалоговое окно.
Save to disk / Load from disk	Текущая конфигурация может быть сохранена в XML файле (*.DUM) и перезагружена из этого файла. Это полезно при установке одинаковой конфигурации пользователей на многих системах. Стандартное диалоговое окно для просмотра в системе файлов предоставляется для этой цели. Фильтр файлов автоматически установлен на *.DUM, что означает файлы device user management управления пользователями устройств.

Распечатка конфигурации управления пользователями

Для распечатки установок вида **Users and Groups** выполните команду **Print** из меню **File** или команду **Document** из меню **Project.**

Права доступа

Обзор

Вид **Access Rights** редактора устройства является частью свойства онлайн управления пользователем *(см. стр. 872)*. Он обслуживает предоставление или запрет определённым в данное время группам пользователей определённых разрешений, таким образом определяя права доступа для пользователей к файлам или объектам (например, к приложению) на контроллере во время работы.

Обратите внимание, что разрешения доступа могут быть назначены только для групп, не для конкретных пользователей. По этой причине Вы должны определить пользователя как члена группы. Производите конфигурацию пользователей и пользовательских групп в виде **Users and Groups** редактора устройства *(см. стр. 872)*.

Рассмотрите пример на следующем изображении: разрешение добавлять и удалять детей к/из объекта **PlcLogic** предоставлено группе пользователей **developpers_1**.

Вид **Access Rights** редактора устройства

Определение разрешений доступа

Для определения разрешения действия для одного или более объектов выполните следующее:

Шаг	Действие
1	Выберите запись объекта или записи ниже желаемого типа действия в области **Actions.**
2	Выберите желаемую группу в области **Permissions.**
3	Щёлкните кнопку **Grant** или **Deny.**

ПРИМЕЧАНИЕ: При установке прав доступа к объекту рассмотрите таблицу разметки, показанную в параграфе *Какое действие в деталях касается определённых прав доступа к определённому объекту (см. стр. 878).*

Смотрите инструкции в области **Howto** вида.

Область Actions (действия)

Область **Actions** перечисляет действия, которые могут быть выполнены во время работы над файлами в файловой системе контроллера или исполняемыми объектами, например, приложениями. Дерево структурировано следующим образом:

На верхнем уровне есть 2 категории объектов, сгруппированных в папки:

- **File system objects** – объекты файловой системы
- **Runtime objects** – выполняемые объекты

Расположенные ниже с отступом категории объектов, являются узлами для 4 типов действий. Они могут быть произведены над конкретными объектами.

- **Modify** (изменение, например, загрузка приложения)
- **View** (мониторинг)
- **Add/remove children** (добавление или удаления дочерних объектов к/из существующего объекта)
- **Execute** (например, старт/стоп приложения, установка точек прерывания)

Ниже каждого типа узла находятся объекты или цели действия. Это файлы контроллера или исполняемые объекты, например **Device.**

Эти записи объектов отображаются древовидной структуре, размечая дерево устройства или стркутуру файловой системы.

ПРИМЕЧАНИЕ: Назначение определения прав доступа родительскому узлу в дереве объектов обычно означает, что дочерние узлы унаследуют это определение. Это справедливо до тех пор пока они не получат своего собственного явного определения. Однако, в зависимости от устройства это может поддерживаться разным образом. В любом случае наследования не визуализируются в этом виде.

Область Permissions - разрешений

Область **Permissions** показывает определённые группы пользователей.

Перед каждой группой одна из следующих иконок указывает назначенное в данный момент разрешение касательно выбранной цели в области **Actions**:

Иконка	Описание
– (знак минуса)	Действия, выбранные в области **Actions**, предоставляются для группы.
+ (знак плюса)	Действия, выбранные в области **Actions**, отвергнуты для группы.
X (знак перекрестия)	Нет определений явных прав доступа для действий, выбранных в области **Actions**
Не отражается иконка	Множество действий выбрано в области **Actions**, которое не имее уникальных установок для выбранной в данной момент группы.

После того как Вы выбрали желаемые объекты ниже желаемых действий в области **Actions** и Вы выбрали желаемую группу в области **Permissions**, Вы можете использовать следующие кнопки:

Кнопка	Описание
Grant	Явное предоставление прав доступа
Deny	Явное исключение прав доступа
Clear	Предоставленные права доступа для действий , выбранных в данный момент в области **Actions** будут удалены, потому что установлены на значения по умолчанию.

Применение и сохранение текущей конфигурации

Кнопки, доступные в верхнем поле диалогового окна:

Кнопка	Описание
Download to device	Загружает сконфигурированные в данный момент определения прав доступа в устройство. Они станут действительными только после загрузки.
Upload from device	Выгрузка прав доступа, применённых в данный момент на устройстве, в диалоговое окно конфигурации.
Save to disk / Load from disk	Текущая конфигурация может быть сохранена в XML файл (*.DAR) и перезагружена из этого файла. Это полезно при установке одинаковой конфигурации пользователей на многих системах. Стандартное диалоговое окно для просмотра в системе файлов предоставляется для этой цели. Фильтр файлов автоматически установлен на *.DAR, что означает файлы device access right –прав доступа к устройствам.

Распечатка определения прав доступа

Для распечатки установок вида **Access Rights** выполните команду **Print** из меню **File** или команду **Document** из меню **Project.**

Какое действие в деталях касается определённых прав доступа к определённому объекту?

Objects			Action	Rights			
				Add/ Remove Children	Execute	Modify	View
Device			Login	–	–	–	X
	Logger		Read entries	–	–	–	X
	PlcLogic						
		Application	Login	–	–	–	X
			Create	–	–	X	–
			Create child	X	–	X	–
			Delete	–	–	X	–
			Download / Online Change	–	–	X	–
			Create bootproject	–	–	X	–
			Read variable	–	–	–	X
			Write variable	–	–	X	X
			Force variable	–	–	X	X
			Set + delete breakpoint	–	X	X	–
			Set next statement	–	X	X	–
			Read callstack	–	–	–	X
			Single cycle	–	X	–	–
			Set flow control	–	X	X	–
			Read flow control	–	–	–	X
			Run / Stop	–	X	–	–
			Reset	–	–	X	–
	Установки		Read settings	–	–	–	X
			Write settings	–	–	X	–
	Управление пользовате-лем		Read configuration	–	–	–	X
			Write configuration	–	–	X	–
X = право должно быть установлено явно – = право не релевантно (не уместно)							

Приложение E

Наборы функций контроллера для миграции

Наборы функций контроллера для миграции

Контроллеры Twido

Контроллер	Dig In	Dig Out	MOD	FC	HSC	PWM	Serial	ETH
TWDLCAA10DRF	6	4	No	3	1	0	1	No
TWDLCDA10DRF	6	4	No	3	1	0	1	No
TWDLCAA16DRF	9	7	No	3	1	0	1+1	No
TWDLCDA16DRF	9	7	No	3	1	0	1+1	No
TWDLCAA24DRF	14	10	No	3	1	0	1+1	No
TWDLCDA24DRF	14	10	No	3	1	0	1+1	No
TWDLCAA40DRF	24	16	No	4	2	2	1+1	No
TWDLCDA40DRF	24	16	No	4	2	2	1+1	No
TWDLCAE40DRF	24	16	No	4	2	2	1+1	Yes
TWDLCDE40DRF	24	16	No	4	2	2	1+1	Yes
TWDLMDA20DTK	12	8	Yes	2	2	2	1+1	No
TWDLMDA20DUK	12	8	Yes	2	2	2	1+1	No
TWDLMDA20DRT	12	8	Yes	2	2	2	1+1	No
TWDLMDA40DTK	24	16	Yes	2	2	2	1+1	No
TWDLMDA40DUK	24	16	Yes	2	2	2	1+1	No

Dig In = число дискретных входов
Dig Out = число дискретных выходов
MOD = модули расширения
FC = число быстрых счётчиков
HSC = число высокоскоростных счётчиков
PWM = число импульсных генераторов
Serial = число последовательных портов
ETH = порты Ethernet

Контроллеры M221

Controller	Dig In	Dig Out	Ana In	MOD	FC	HSC	PWM	Serial	ETH	CART
TM221C16R	9	7	2	TM2/ TM3	4	2	0	1	No	1
TM221C16T	9	7	2	TM2/ TM3	4	2	2	1	No	1
TM221C24R	14	10	2	TM2/ TM3	4	2	0	1	No	1
TM221C24T	14	10	2	TM2/ TM3	4	2	2	1	No	1
TM221C40R	24	16	2	TM2/ TM3	4	2	0	1	No	2
TM221C40T	24	16	2	TM2/ TM3	4	2	2	1	No	2
TM221CE16R	9	7	2	TM2/ TM3	4	2	0	1	Yes	1
TM221CE16T	9	7	2	TM2/ TM3	4	2	2	1	Yes	1
TM221CE24R	14	10	2	TM2/ TM3	4	2	0	1	Yes	1
TM221CE24T	14	10	2	TM2/ TM3	4	2	2	1	Yes	1
TM221CE40R	24	16	2	TM2/ TM3	4	2	0	1	Yes	2
TM221CE40T	24	16	2	TM2/ TM3	4	2	2	1	Yes	2
TM221M16R/G	8	8	2	TM2/ TM3	4	2	0	2	No	0
TM221M16T/G	8	8	2	TM2/ TM3	4	2	2	2	No	0
TM221M32TK	16	16	2	TM2/ TM3	4	2	2	2	No	0
TM221ME16R/G	8	8	2	TM2/ TM3	4	2	0	1	Yes	0
TM221ME16T/G	8	8	2	TM2/ TM3	4	2	2	1	Yes	0
TM221ME32TK	16	16	2	TM2/ TM3	4	2	2	1	Yes	0

Dig In = число дискретных входов
Dig Out = число дискретных выходов
Ana In = число аналоговых входов
MOD = модули расширения
FC = число быстрых счётчиков
HSC = число высокоскоростных счётчиков
PWM = число импульсных генераторов
Serial = число последовательных портов
ETH = порты Ethernet
CART = число картриджей

Контроллеры SoMachine

Controller	Dig In	Dig Out	Ana In	MOD	FC	HSC	PWM	Serial	ETH	CART
TM241C24R	14	10	0	TM2/ TM3	4	2	2	2	No	1
TM241C24T/U	14	10	0	TM2/ TM3	4	2	2	2	No	1
TM241C40R	24	16	0	TM2/ TM3	4	2	2	2	No	2
TM241C40T/U	24	16	0	TM2/ TM3	4	2	2	2	No	2
TM241CE24R	14	10	0	TM2/ TM3	4	2	2	2	Yes	1
TM241CE24T/U	14	10	0	TM2/ TM3	4	2	2	2	Yes	1
TM241CE40R	24	16	0	TM2/ TM3	4	2	2	2	Yes	2
TM241CE40T/U	24	16	0	TM2/ TM3	4	2	2	2	Yes	2
TM241CEC24R	14	10	0	TM2/ TM3	4	2	2	2	Yes	1
TM241CEC24T/U	14	10	0	TM2/ TM3	4	2	2	2	Yes	1
HMISCU•A5	16	10	0	No	2	1	2	1	Yes	0
HMISCU•B5	8	8	2	No	2	1	2	1	Yes	0

Dig In = число дискретных входов
Dig Out = число дискретных выходов
Ana In = число аналоговых входов
MOD = модули расширения
FC = число быстрых счётчиков
HSC = число высокоскоростных счётчиков
PWM = число импульсных генераторов
Serial = число последовательных портов
ETH = порты Ethernet
CART = число картриджей

Словарь

A

application

Программа, включая данные конфигурации, символы, и документацию.

C

CFC

(*continuous function chart*) Графический язык программирования (расширения стандарта IEC 61131-3) основанный на языке схем функциональных блоков, который работает как блок-схема. Однако не требуется цепей и возможно свободное размещение блоков, что разрешает контуры обратной связи. Для каждого блока входы расположены слева и выходы справа. Вы можете соединять выходы блока с входами других блоков для создадия сложных выражений.

configuration

расположение и взаимное соединение компонентов аппаратных средств в системе и параметры аппаратных и программных средств, которые определяют рабочие характеристики системы.

controller

Автоматизирует промышленные процессы (также известен как программируемый логический контроллер или программируемый контроллер).

D

DTM

(*device type manager – менеджер типа устройства*) Классифицируются на две категории:

- Device DTMs соединяют компоненты конфигурации полевых устройств.
- CommDTMs соединяют коммуникационные компоненты программного обеспечения.

DTM обеспечивает унифицированную структуру для доступа к параметрам устройства и конфигурирования, действий и диагностирования устройств. DTM могут иметь диапазон от простого графического интерфейса пользователя для установки параметров устройства, до высоко продуманных приложений, способных производить сложные вычисления в реальном времени для для целей диагноза и обслуживания.

DUT

(*data unit type*) Наряду со стандартными типами данных пользователь может определять собственные типы структур, типы перечислений и ссылки как элементы типов данных в редакторе DUT.

E

element

Короткое имя элемента ARRAY.

expansion bus

Электронная коммуникационная шина между модулями расширения Вх/Вых и контроллером.

F

FBD

(*function block diagram*) Один из 5 языков для программирования логики или управления, поддерживаемый стандартом IEC 61131-3 для систем управления. Диаграммы функциональных блоков – графически ориентированный язык программирования. Он работает со списком цепей, где каждая цепь содержит графическую структуру из блоков и соединительных линий, что представляет или логическое, или арифметическое выражение, вызов функционального блока, переход, или инструкцию возврата

FDT

(field device tool) Спецификация, описывающая стандартизованный обмен данными между устройствами и системой управления или инструментами управления, или инжинирингом, или активами.

G

GRAFCET

Функционирование последовательного действия в структурированной и графической форме.

Это аналитический метод, который разделяет любую систему последовательного управления на серию шагов с которыми связаны действия, переходы и условия.

GVL

(*global variable list*) Управляет глобальными переменными в проекте SoMachine.

I

I/O

(*input/output*) входы/выходы

IL

(*instruction list*) Программа, написанная на языке, который состоит из серии текстовых инструкций, выполняемых последовательно контроллером. Каждая инструкция содержит номер линии, код инструкции и операнд (обращайтесь к IEC 61131-3).

L

LD

(*ladder diagram*) Графическое представление инструкций программы контроллера с помощью символов контактов, катушек и блоков в серии цепей, выполняемых последовательно контроллером (обращайтесь к IEC 61131-3).

M

MAC address

(*media access control address*) Уникальное 48-битное число, связанное со специфической частью аппаратных средств. MAC адрес программируется в каждой сетевой карте или устройстве при их производстве.

P

POU

(*program organization unit*) Декларация переменных в исходном коде и соответствующий набор инструкций. POUs облегчает модульное повторное использование программ, функций и функциональных блоков. Однажды объявленные, POUs доступны одна другой.

program

Компонент приложения, который состоит из скомпилированного исходного кода, способного быть установленным в памяти логического контроллера.

R

RTS

(*request to send*) Запрос на посылку. Сигнал передачи данных и сигнал CTS, который подтверждает RTS от узла получателя.

S

Sercos

(*serial real-time communications system*) Цифровая шина управления, которая соединяет между собой контроллеры движения, электроприводы, Вх/Вых, датчики и исполнительные устройства для машин и систем с цифровым управляением.

SFC

(*sequential function chart*) Язык, который состоит их шагов со связанными действиями, переходами со связанными логическими условиями и прямыми связями между шагами и переходами. (Стандарт SFC определён в IEC 848. Он совместим с IEC 61131-3)

U

UDP

(*user datagram protocol*) Протокол без режима соединения (определён в IETF RFC 768) в котором сообщения разделяются на датаграммы (телеграммы данных) к принимающему компьютеру по IP сети. UDP протокол обычно связывается с Internet протоколом UDP/IP, сообщения не ожидают ответа и, следовательно, идеальны для приложений в которых пропущенные пакеты не требуют повторной передачи (такие как потоковое видео и сети, которые требуют производительности реального времени).

Ссылки

Символы

IronPython, *823*
__DELETE
operator, *695*
__ISVALIDREF
operator, *698*
NEW
operator, *699*
__QUERYINTERFACE
operator, *702*
__QUERYPOINTER
operator, *704*

0-9

3 GB switch on Windows 7 32-bit, *781*
32-bit operating system memory limit, *781*

A

ABS
IEC operator, *681*
ACOS
IEC operator, *690*
ADD
IEC operator, *619*
adding controllers, *64*
adding controllers by drag and drop, *60* adding devices and modules by drag and drop, *61*
adding devices from device template by drag and drop, *62*
adding devices from function template by drag and drop, *62*
adding expansion devices by drag and drop, *61*
addressing, *801*
ADR
operator, *659*
analog inputs CANopen, *777*
AND
IEC operator, *633*
ANY_NUM_TO
IEC operator, *679*
ANY_TO
IEC operator, *679*
ASIN
IEC operator, *689*
ATAN
IEC operator, *691*
automatic I/O mapping, *137*

B

BIT, *584*
BITADR
operator, *661*
BOOL, *579*
BOOL_TO
IEC operator, *665*
boot application, *232*
Boot Application, *225*
build-time performance, *782*

C

CAL
IEC operator, *662*
CANopen analog inputs, *777*
CANopen devices, *66*
CASE
instruction, *358*
Catalog view, *43*
command
Convert Device, *73*
Convert SoMachine Basic Project, *77*
Convert Twido Project, *77*
communication manager configuration, *66*
communication settings, *100*

configuration diagnostic, *69*
constants, *520*
Content
operator, *660*
CONTINUE
instruction, *358*
Convert Device command, *73* convert SoMachine Basic project, *77* convert Twido project, *77*
COS
IEC operator, *687*

D

data types, *579*
DATE, *581*
DATE_AND_TIME, *581*
DATE_TO
IEC operator, *674*
device templates, *745*, *746*
devices
adding, *68*
DIV
IEC operator, *625*
download, *224*, *232*
boot application, *232*
Boot Application, *225*
DT, *581*
DT_TO
IEC operator, *674*
DTM, *38*

E

EQ
IEC operator, *656*
EXIT
instruction, *358*
EXP
IEC operator, *685*
expansion devices, *65*
expansions, *65*
EXPT
IEC operator, *692*
external variables, *517*

F

FB_init
IEC operator, *708*
FdtConnections node, *38*
FFB Finder, *422*
field device configuration, *66*
fieldbus health information, *68* fieldbusses supported by templates, *735* FOR
instruction, *358*
Function and Function Block Finder, *422*
function templates, *759*

G

GE
IEC operator, *655*
global variables, *516*
GNVL
global network variables list, *391*
GT
IEC operator, *652*

H

health information fieldbus, *68*
HMI controller
unsuccessful multiple download, *784*

I

IEC objects
fieldbus Diagnostic/O mapping, *68*
IF
instruction, *358*
increasing build-time performance, *782*
INI
IEC operator, *708*
input and output variables, *515*
input variables, *514*
installation
third-party Sercos devices, *57*

instructions
ST editor, *358*
INT_TO
IEC operator, *669*
integer, *579*

J

JMP
instruction, *358*

L

large SoMachine projects, *781*
LE
IEC operator, *654*
LIMIT
IEC operator, *649*
literals
typed, *521*
LN
IEC operator, *683*
local variables, *514*
LOG
IEC operator, *684*
login, *224*, *232*
LREAL, *580*
LREAL_TO
IEC operator, *670*
LT
IEC operator, *653*
LTIME, *583*

M

managing tags, *43*
MAX
IEC operator, *647*
memory consumption of SoMachine, *782*
memory limit, *781*
menus, *779*
MIN
IEC operator, *648*
MOD
IEC operator, *628*
Modbus IOScanner on a Serial Line, *783*
Modbus Serial IOScanner error detected, *783*
Modbus SL devices, *66*
Modbus slave disconnected, *783*
MOVE
IEC operator, *630*
MUL
IEC operator, *621*
multiple download not working on HMI controller, *784*
MUX
IEC operator, *650*

N

NE
IEC operator, *657*
Network Device Identification accessing new controllers, *787*
connecting via IP address and address information, *789*
FAQs, *792*
in Controller Selection view, *97*
NodeName, *102*
NOT
IEC operator, *636*
NVL
configuration example, *396*
considerations, *387*
controllers supporting NVL, *387* network variables list, *386* rules, *393*
NVL communication suspended, *784*

O

OPC client
variables mapped to %I , *812*
OPC server, *808*
OPC Server 3, *807*
OPC server configuration, *813*
OR
IEC operator, *634*
output variables, *515*

P

persistent variables, *519*
Process communication settings dialog box, *100*
programming environments for Python, *823*
publishing variables, *486* publishing variables (HMI), *489* Python, *823*

R

REAL, *580*
REAL_TO
IEC operator, *670*
reducing memory consumption, *782*
remanent variables, *518*
REPEAT
instruction, *358*
retain variables, *518*
RETURN
instruction, *358*
ROL
IEC operator, *641*
ROR
IEC operator, *643*
routing, *801*
run, *242*

S

Script Engine, *823*
searching within catalogs, *43*
SEL
IEC operator, *646*
selecting variables, *488*
Sercos
installation of third-party devices, *57*
SHL
IEC operator, *638*
shortcuts, *779*
SHR
IEC operator, *640*
SIN
IEC operator, *686*
SIZEOF
IEC operator, *631*
smart coding, *575*
SQRT
IEC operator, *682*
ST editor
instructions, *358*
startup performance of SoMachine, *778*
static variables, *516*
stop, *242*
STRING, *581*
STRING_TO
IEC operator, *676*
SUB
IEC operator, *623*
symbol configuration, *478*

T

tagging catalog items, *43*
TAN
IEC operator, *688*
tasks
adding, *221*
template libraries, *745*
templates, *732*
temporary variables, *516*
TIME, *581*
time data types, *581*
TIME_OF_DAY, *581*
IEC operator, *672*
TIME_TO
IEC operator, *672*
TO_BOOL
IEC operator, *667*
TOD, *581*
TRUNC
IEC operator, *678*
TRUNC_INT
IEC operator, *679*
typed literals, *521*

U

UNION, *583*

updating devices, *71*
Use DTM Connection checkbox, *38*
user management, *123*
Users and Groups, *123*

V

variable definition, *482*
variable types, *482*
variables, *514*
persistent, *205*
publishing, *486*
publishing (HMI), *489*
remanent, *205*
visualizations, *750*

W

WHILE
instruction, *358*
WSTRING, *584*

X

XOR
IEC operator, *635*