Разработка библиотек APROL
TM860

Усовершенствование процесса автоматизации

www.br-automation.com

Необходимые компоненты

Учебные программы:	TM830 - Разработка проектов APROL
Программное обеспечение:	***APROL*** 2.2 или более поздняя версия
	Automation Runtime 2.66 или
	более поздняя версия
	SuSE Linux 8.2 или более поздняя версия
Аппаратое обеспечение:	1 сервер, 1 x контроллер

Оглавление

1. ВВЕДЕНИЕ

Компания B&R в рамках системы ***APROL*** предоставляет библиотеки, доступные для использования в любое время, расширяемые и соответствующим образом поддерживаемые. Однако, не гарантируется реализация всех функций, относящихся к проекту и перечисленных в листе спецификаций. Часто встречаются очень специфические требования.

Для реализации подобных требований всегда можно создать отдельную библиотеку. При необходимости данную библиотеку можно использовать для воспроизведения логических операций и визуализации.

В следующих разделах объясняется способ настройки собственных библиотек.

Рис.1. Все ли требования учтены?

1.1 Цели

Целью настоящего курса обучения является демонстрация всех этапов создания библиотеки. После завершения курса обучающиеся смогут создавать и связывать множество логических операций различных типов с приложениями визуализации.

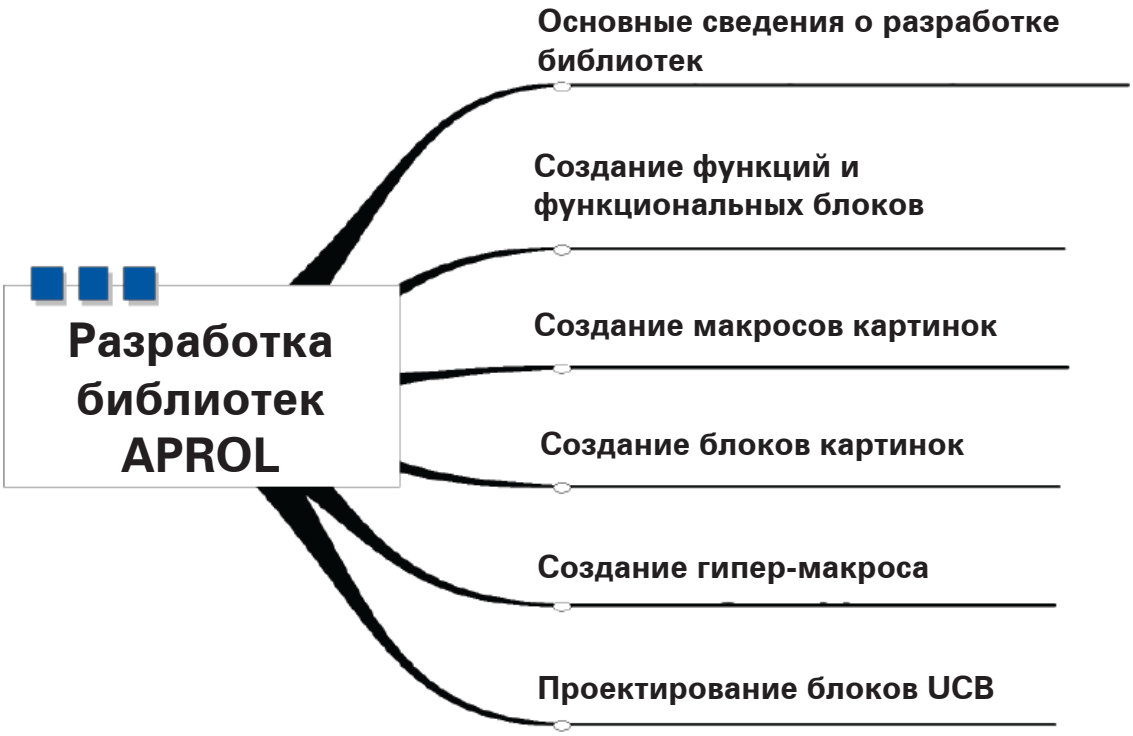


Рис. 2. Общее представление

2. ОСНОВНЫЕ СВЕДЕНИЯ О РАЗРАБОТКЕ БИБЛИОТЕК

Создание новой библиотеки по конкретному проекту.

2.1 Общая информация

Помимо библиотек, поставляемых вместе с системой ***APROL***, допускается создание отраслевых библиотек. Необходимо отметить, что вышеуказанные библиотеки и функциональные блоки могут создаваться и редактироваться проектировщиками, обладающими соответствующими правами в рабочей области разработки библиотеки. Назначение соответствующих прав производится в рабочей области управления проектировщиками (см. введение в систему APROL).

Функциональные блоки являются объектами с таким же доступом и механизмом создания версий, как и другие части проекта (структурные схемы, диаграммы процессов и т.д.).

В последующих разделах объясняется, как создавать функциональные блоки различных типов на базе имеющихся. Также будут затронуты вопросы тестирования разработанных блоков. Еще один важный раздел настоящей документации посвящен макросам анимационных изображений.

Могут быть созданы функциональные блоки следующих типов:

Тип функционального блока	Описание
Макросы изображений	Графическое объекты с анимированным контентом
Функциональные блоки видеодрайверов	В сочетании с макросом изображений используются для приложений визуализации
Блоки логических функций	Код на языке C, функции и функциональные блоки
Функциональные блоки UCB	Программирование на языке скриптов Linux и Python
Функциональные блоки систем управления	Функциональные блоки, разработанные компанией B&R (сигнализация, тренд и т.д.)
Гипер-макросы	Сочетание всех функциональных блоков

2.2 Стандартные библиотеки

Следующие библиотеки поставляются вместе с системой ***APROL***.

STANDARD	Содержит все ***функциональные блоки в соответствии с EC 7137-3***, а также функциональные блоки систем управления для настройки ***систем трендов и сигнализации*** в ***APROL***.
DCS	Содержит ***гипер-макросы*** и ***стандартные элементы*** для ***приводов, фурнитуры, регуляторов*** и ***измеренных значений***.
SYSTEM	Содержит функциональные блоки и гипер-макросы, позволяющие настраивать ***самодиагностику системы***.
ISO70628	Содержит технологические карты для систем производственных процессов.
SYSTEM_RIO	Используется для мониторинга удаленных устройств I/O компании B&R.
mSyS	Содержит mSyS.Report и mSyS.Service.

При настройке структурных схем имеющиеся в библиотеках функциональные блоки используются вместе с графическим языком функциональных блоков и связей (IEC 1131-3).

Библиотеки, включенные в поставку ***APROL***, содержат группы полностью запрограммированных, проверенных и задокументированных функциональных блоков. За счет обращения к заранее определенным символам и функциональным возможностям удается упростить проектно-конструкторские работы.

2.3 Создание библиотеки

Чтобы создать новую **библиотеку**, в первую очередь необходимо закрыть библиотеку или проект при помощи пункта меню **File / Close (Файл/ Закрыть)**. Затем выберите в CaeManager **папку Libraries (библиотеки)**. Выберите пункт меню **File / New (Файл/ Новый)**, чтобы открыть окно **New library (новая библиотека)** .

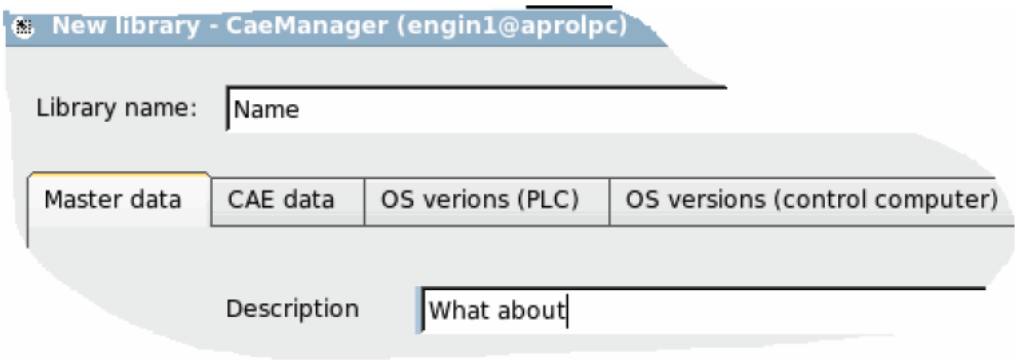


Рис. 3. Создание новой библиотеки

В окне вводится **имя библиотеки**. После создания библиотеки можно открыть окно свойств для изменения параметров по умолчанию (**File / Properties (Файл/ Свойства) или щелчок правой кнопки мыши открытой библиотеки → Properties (свойства)**).

2.3.1 Основные данные библиотеки

На вкладке основных данных **master data** по существу отображаются описание и тип библиотеки **(CUSTOMER / PARTNER / B&R)**. Все остальные записи генерируются автоматически.

На вкладке можно ввести или отредактировать короткое описание библиотеки.

В ***APROL*** выделяются три типа библиотек: **CUSTOMER, PARTNER** и **B&R**. Между ними существуют следующие отличия:

Тип библиотеки	Описание
CUSTOMER	Пользовательские библиотеки могут быть считаны и изменены любым проектировщиком, обладающим соответствующими правами на библиотеку.
PARTNER	Библиотеки партнеров могут быть защищены паролем. Пользователи, обладающие всеми правами на библиотеки, получают права доступа на чтение всей информации, значимой для проектирования проекта, без требования пароля. Не могут выбираться вкладки **Declaration (объявление)** и **Program (программа)**. Пароль может потребоваться для обеспечения доступа на запись (для внесения изменения на вкладках **Declaration** и **Program**).
B&R	Тип библиотеки "B&R" обычно выводится серым цветом, что означает возможность создания библиотек данного типа только сотрудниками компании B&R. Библиотеки B&R могут считываться и модифицироваться только при входе с паролем администратора B&R.

Тип библиотеки можно выбрать при помощи переключателя с соответствующим именем. Параметр "Set password" (задать пароль) включается только при выборе создания библиотеки партнеров. Ввод необходимо подтвердить при помощи [OK].

2.3.2 Данные CAE для библиотеки (управление версиями)

Так же, как и при разработке проекта (см. введение в систему APROL), при проектировании библиотек можно определить **количество версий** блока, которые необходимо хранить. Это же справедливо и для **подтвержденных версий**.

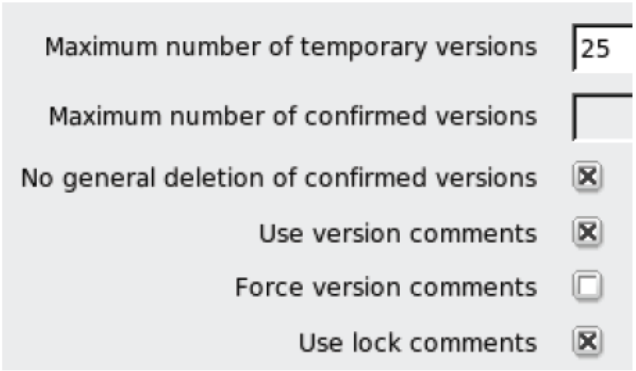


Рис. 4. Данные CAE библиотеки (управление версиями)

Выбор параметра **No general deletion of confirmed versions** препятствует удалению подтвержденных версий объектов библиотеки.

Maximum number of temporary versions: в поле указывается количество версий, которое необходимо хранить в базе данных (механизм кольцевого буфера).

2.3.3 Версии ОС (контроллера) для библиотеки

На вкладке **OS version** отображается **операционная система контроллера** (контроллер компании B&R). Т.к. компания B&R поставляет и контроллеры Intel, и контроллеры Motorola, существенно отличающиеся по версиями ОС, то на вкладке можно настроить данный параметр. После настройки будет генерироваться код только для введенной версии операционной системы.

Также на вкладке можно определить зависимости от библиотек B&R Automation Studio или пользовательских библиотек B&R Automation Studio.

Для этого нажмите на кнопку **Dependencie**. Могут быть включены зависимости от дополнительных библиотек или модулей. **Функции** или **функциональные блоки** добавленных библиотек могут быть выполнены при вызове их из другого **функционального блока**.

Рис. 5. Выбор среды компиляции

2.3.4 Версии ОС (управляющего компьютера) для библиотеки

Вкладка относится к операционной системе используемого компьютера. Снятие флага **SuSE** приводит к **отсутствию генерации кода на уровне управляющего компьютера для данной библиотеки**. Это означает, что функции и функциональные блоки, созданные для этой **библиотеки**, могут использоваться только на контроллере (используя параметры вкладки OS versions (controller)). При установке **флага SuSE** допускается, при необходимости, задание определенных зависимостей (библиотек LINUX). Операция выполняется так же, как и указание зависимостей контроллера.

2.4 Создание групп функциональных блоков

Для упрощения просмотра библиотек **функциональные блоки** рассортированы по **группам функциональных блоков**. Перед созданием в библиотеке функционального блока **предварительно должна быть создана по крайней мере одна группа функциональных блоков**.

Группам функциональных блоков необходимо дать логическое имя. Например, имя группы должно в некоторой степени отражать выполняемые функции и типы входящих в нее функциональных блоков.

После открытия соответствующей библиотеки можно создать новую группу функциональных блоков при помощи пункта меню **Create part / Group (Создать элемент/ Группа)**. В соответствующие поля вводятся имя и описание новой группы.

Рис. 6. Создание новой группы функциональных блоков

2.5 Общая информация о создании функциональных блоков

Чтобы упростить ориентирование и обеспечить высокую степень открытости при навигации по библиотекам и их группам функциональных блоков, каждому **типу функциональных блоков присвоено условное обозначение**.

Ниже на рисунке показаны все типы **функциональных блоков** и соответствующие им условные обозначения.

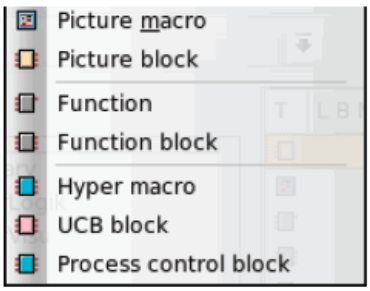


Рис. 7. Типы функциональных блоков

Тип функционального блока	Описание
Макрос изображений	Графический объект с анимационной информацией. Макрос изображений впоследствии назначается функциональному блоку видеодрайвера.
Функциональный блок видеодрайвера	Интерфейс между логическим компонентом и диаграммой процесса. Функциональные блоки видеодрайверов поддерживают макросы изображений. Цвет на структурной схеме: желтый/ коричневый.
Функция	Логический компонент, созданный на языке ANSI C, который обрабатывается за один цикл и чье возвращаемое значение (выход) называется точно так же, как и сама функция. Цвет на структурной схеме: серый.
Функциональный блок	Логический компонент, созданный на языке ANSI C, который может возвращать одно или несколько значений (выходных значений). Локальные переменные внутри функционального блока сохраняют свои значения. Цвет на структурной схеме: серый.
Функциональный блок системы управления	Компания B&R Industrie-Elektronik GmbH предоставляет функциональные блоки систем управления, охватывающие большинство функций подобных систем, таких как настройка систем сигнализации и передачи сообщений.
Блок UCB	Функции, для которых не существует стандартных типов. В большинстве случаев подобные блоки создаются при помощи программирования на языке оболочки (программирования на языке скриптов LINUX). Цвет на структурной схеме: светло-красный.
Гипер-макрос	Структурные схемы и макросы, которые можно размещать как функциональные блоки. Разъемы на схемах I/O выходят из функционального блока в виде штыревых контактов и могут встраиваться в окружающую логику. Помимо этого существуют константы гипер-макросов, которые могут использоваться для настройки функционального блока. Цвет на структурной схеме: тесно-синий.

После создания группы функциональных блоков можно выбрать пункт меню **Create part / Function (Создать элемент/ Функция)**.

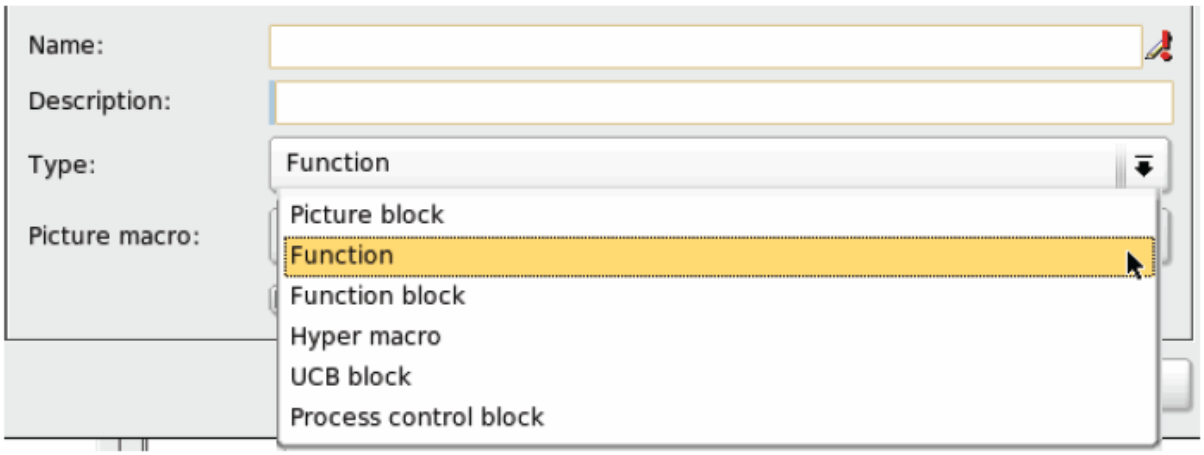


Рис. 8. Создание новой функции

Записи после **Create in (создать в)** и следующие за **Description (описание)** берутся из выбранной библиотеки и сопутствующего ей описания.

Остальные поля необходимо заполнить следующим образом:

Group (группа) — Выбирается группа функциональных блоков, в которой необходимо создать новую функцию.

Type (тип) — Выбирается тип функционального блока (см. список на рис. 7).

Picture macro (макрос картинок) — При создании **блока картинок** можно выбрать сопровождающий макрос, который представляется данным блоком.

Remanent (остаточный) — При создании **функционального блока** параметр **Remanent** включается по умолчанию. Обычно флажок определяет действия по отношению к каким-либо сохраненным данным. В области остаточной памяти хранятся значения даже после инициализации контроллера, задача заключается в повторной загрузке или повторном запуске системы управления процессом.

Если флажок **Remanent** сброшен, эти значения будут потеряны.

При нахождении в **группе** параметры применяются автоматически. Также допускается создание функционального блока без предварительного выбора группы функциональных блоков. В подобном случае поле **Group (группа)** можно редактировать.

Записи и выбор параметров необходимо подтвердить при помощи **[OK]**. Затем **"пустой" функциональный блок** помещается в библиотеку.

Для последующей разработки новый функциональный блок можно открыть в соответствующей группе в CaeManager.

Отдельные действия, которые необходимо совершить для разработки функционального блока, показываются в библиотеке CAE на различных вкладках, соответствующих различным типам отображения. Два рядом расположенных окна позволяют показывать одновременно два различных представления.

Иногда полезно ограничить представление одним типом отображения. Для скрытия окна справа можно использовать пункт меню **View** / **Fullscreen (Вид/ Полный экран)**. При этом окно слева расширяется, упрощая ввод данных в области ввода (например, при записи документации к функциональному блоку). **Полноэкранное представление** также можно вызвать при помощи **соответствующей кнопки на панели инструментов**.

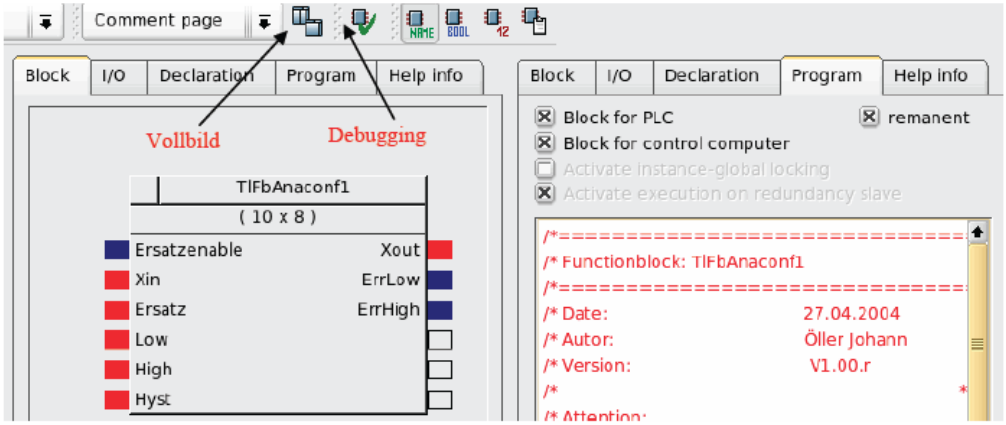


Рис. 9. Вкладки "Block" (блок) и "Program" (программа)

2.6 Созданная библиотека

В данный момент новая библиотека создана и помещена в группу. При необходимости допускается расширение группы, либо можно приступить к фактической разработке библиотеки.

Библиотека

Рис. 10. Добавленная библиотека

Упражнение 1:

Создать новую библиотеку и группу в соответствии с вашими предпочтениями. В процессе обучения будут реализованы очень разноплановые блоки.

3. СОЗДАНИЕ ФУНКЦИЙ И ФУНКЦИОНАЛЬНЫХ БЛОКОВ

APROL предоставляет несколько стандартных функций. Несмотря на это, часто требуется создание пользовательских функциональных блоков. Это можно сделать в любое время в пользовательской библиотеке.

3.1 Создание функции или функционального блока

Пункт меню **Create part / Function (Создать элемент/ Функция)** можно использовать для создания новой функции в группе, принадлежащей библиотеке. Функциональному блоку необходимо дать имя в соответствующем окне ввода.

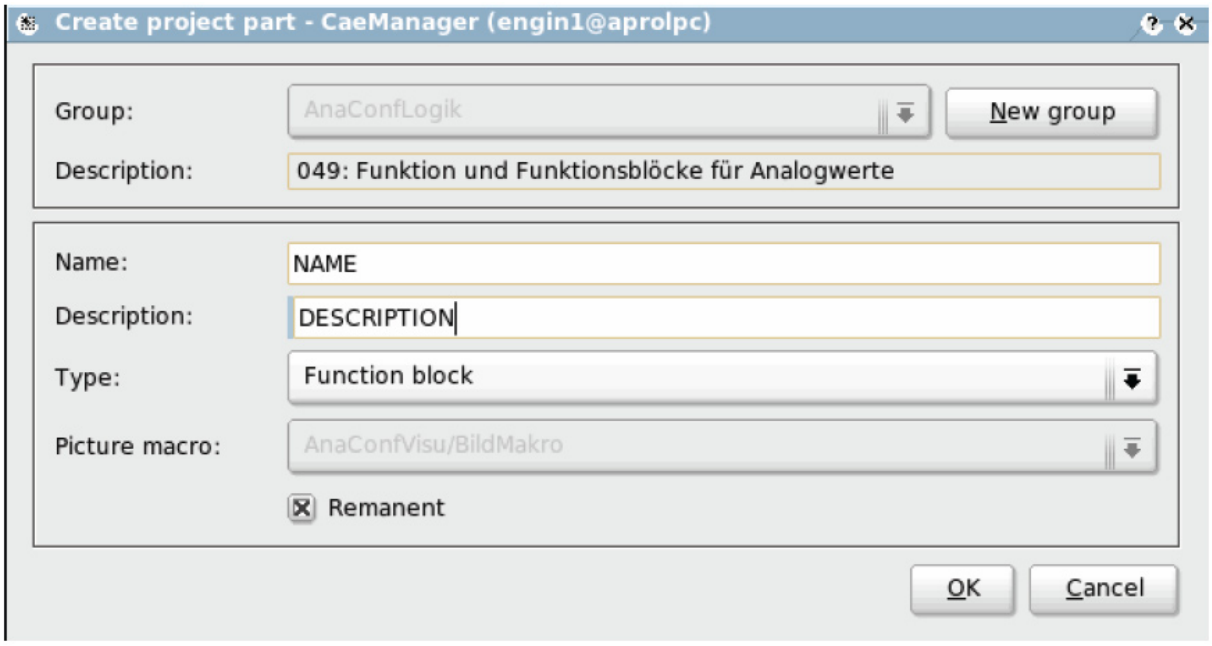


Рис. 11. Создание функционального блока

В зависимости от логики, которую необходимо реализовать, выберите либо **function (функция)** либо **function block (функциональный блок)**. Данную программу можно открыть и редактировать в правой части CaeManager, дважды щелкнув ее.

При этом становятся доступны следующие вкладки:

Имя	Описание
Блок	Создание входных и выходных значений функции
I/O	Значения по умолчанию, права, типы данных и т.д.
Declaration	Внутренние переменные и массивы, заголовки, определения, включения и т.д.
Program	Код программы на языке C (ANSI C)
Help info	Документация на функцию в виде текста или HTML

3.2 Представление блоков при проектировании функциональных блоков

Сопоставление входа/ выхода **функционального блока** может быть сделано в **представлении блока**. Для этого щелкните правой кнопкой мыши блок и выберите **New pin (новый контакт)**. Контакт можно редактировать и именовать, щелкнув по нему.

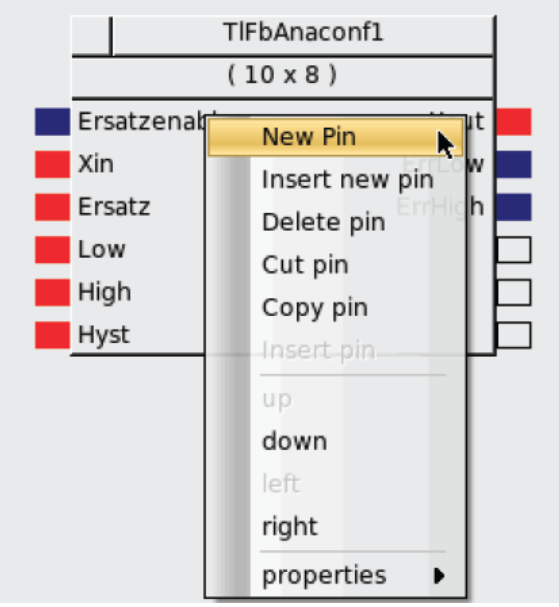


Рис. 12. Создание нового контакта

Тип данных контакта можно изменить, **щелкнув по контакту правой кнопкой мыши**, например, выбрав **контакт, выделенный синим цветом**. При этом открывается список возможных типов данных, из которых следует выбрать подходящий.

3.3 Представление I/O при проектировании функциональных блоков

Для изменения типа данных контакта также можно использовать **представление I/O**. Помимо этого, в представлении можно определить **значения по умолчанию для входных параметров функционального блока**. Это важно, т.к. код программы

генерируется **APROL** на языке ANSI C, для которого указанные значения являются обязательными. Компилятор выдаст ошибку, если не назначены значения по умолчанию или на контакт не подано питание на этапе разработки проекта.

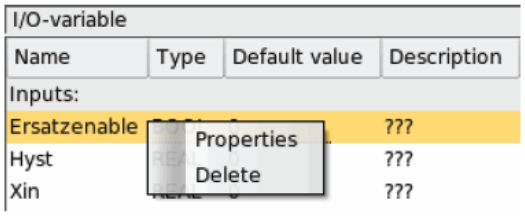


Рис. 13. Свойства I/O (контакт)

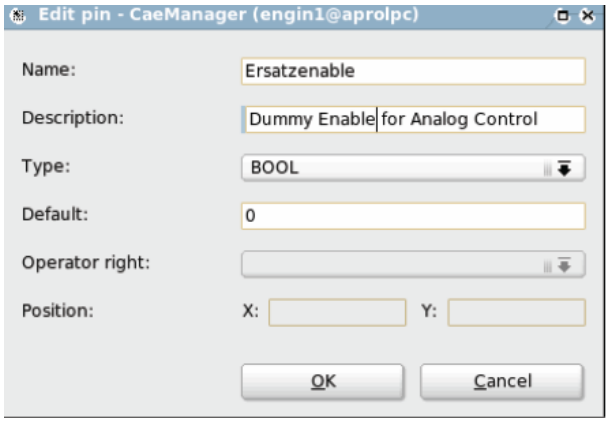


Рис. 14. Изменение I/O (контакта)

Редактирование в основном выполняется так же, как для блока картинок, а не **функционального** блока. Единственное отличие заключается в том, что для выходных параметров могут быть указаны дополнительные права оператора и открытые позиции для макроса.

3.4 Представление объявлений при проектировании функциональных блоков

Общие заголовки, включения и определения могут указываться в представлении определений. Также могут быть созданы локальные переменные (например, переменные циклов), с помощью которых можно реализовать массивы. Конечно, локальные переменные могут быть также созданы непосредственно в коде программы на C. Однако это неудобно, т.к. созданные таким способом переменные не отображаются в процессе отладки библиотеки.

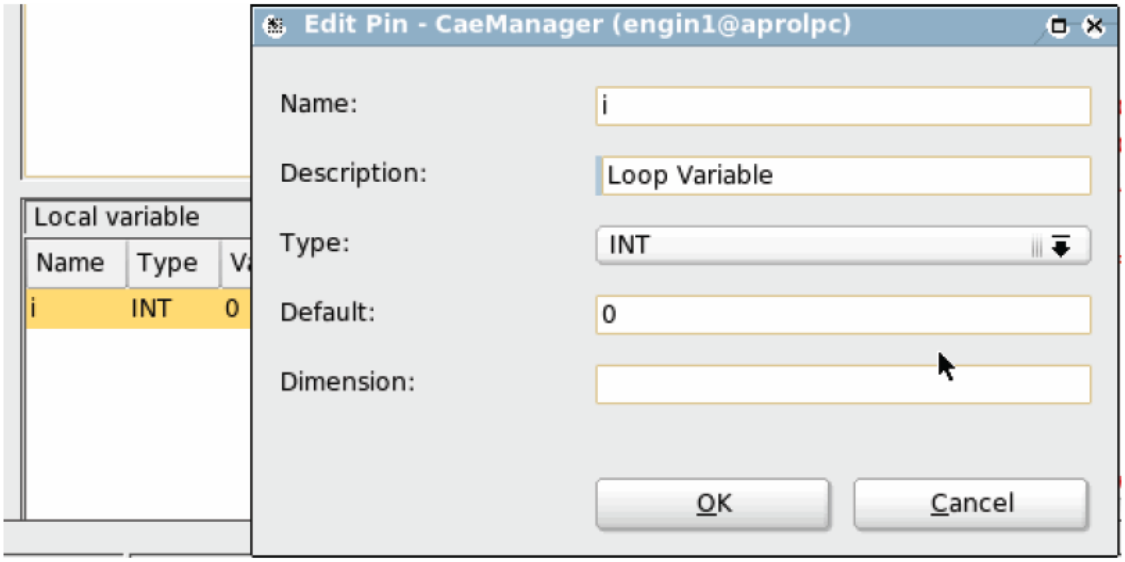


Рис. 15. Объявление внутренних переменных

Включения упрощают интеграцию библиотек B&R с Automation Studio. Выполняя вызовы библиотек, программист не тратит время на повторное написание кода с нуля. Примеры приведены в курсе углубленного изучения ***APROL***.

3.5 Представление программ при проектировании функциональных блоков

В этом представлении вводится простой для понимания **код на C**, снабженный комментариями.

Для кода применяются **правила ANSI C**.

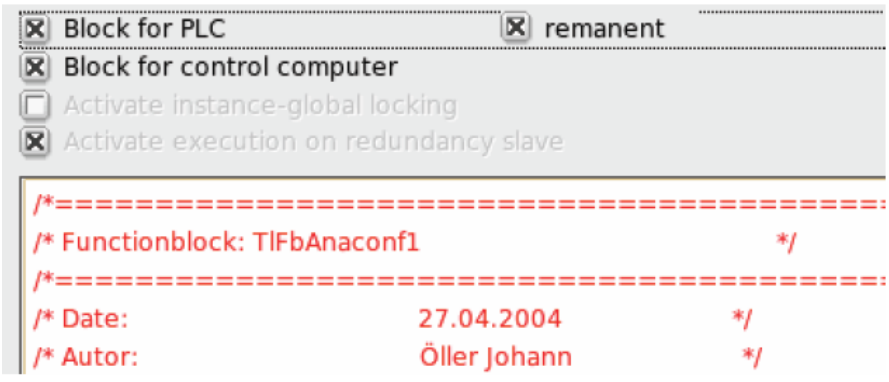


Рис. 16. Окно кода на C (полноэкранный режим)

Подробной информации о программировании на языке ANSI C не приводится, т.к. это выходит за рамки текущего курса обучения. Простые конструкции могут реализовываться практически так же, как и на других языках высокого уровня (например, структурированный текст).

3.6 Представление справочной информации при проектировании функциональных блоков

Представление справочной информации может использоваться для хранения **документации** на новые функциональные блоки или предоставлять **ссылку на HTML-файл**. Лучшим и более понятным способом предоставления документации является **HTML-файл**.

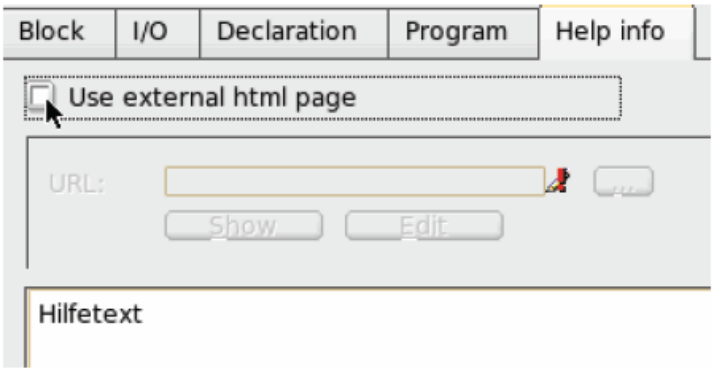


Рис.17. Окно справочной информации (ссылка)

В дальнейшем **документация** по этому блоку доступна **в любой точке системы.**

3.7 Отладка при проектировании функциональных блоков

После реализации **функционального блока** его работа может быть **проверена** непосредственно в библиотеке при помощи **отладчика**. Однако перед отладкой блок необходимо сохранить и скомпилировать. Отладка включается из пункта меню **View / Debug (Вид/ Отладка)** или при помощи значка ярлыка на панели инструментов.

Рис. 18. Включение/ отключение отладки

После включения отладки блок повторно компилируется. После завершения компиляции открывается новый пользовательский интерфейс, показывающий **входные и выходные параметры блока**. Данный пользовательский интерфейс позволяет назначать соответствующие значения и запускать циклическое выполнение. Это может быть выполнено для моделирования циклов, которые будут выполняться контроллером или задачей управляющего компьютера.

Также можно открыть **исходный код** или любую созданную **локальную переменную** для анализа внутренней работы функционального блока. Цикл можно изменять, а также выполнять код по шагам.

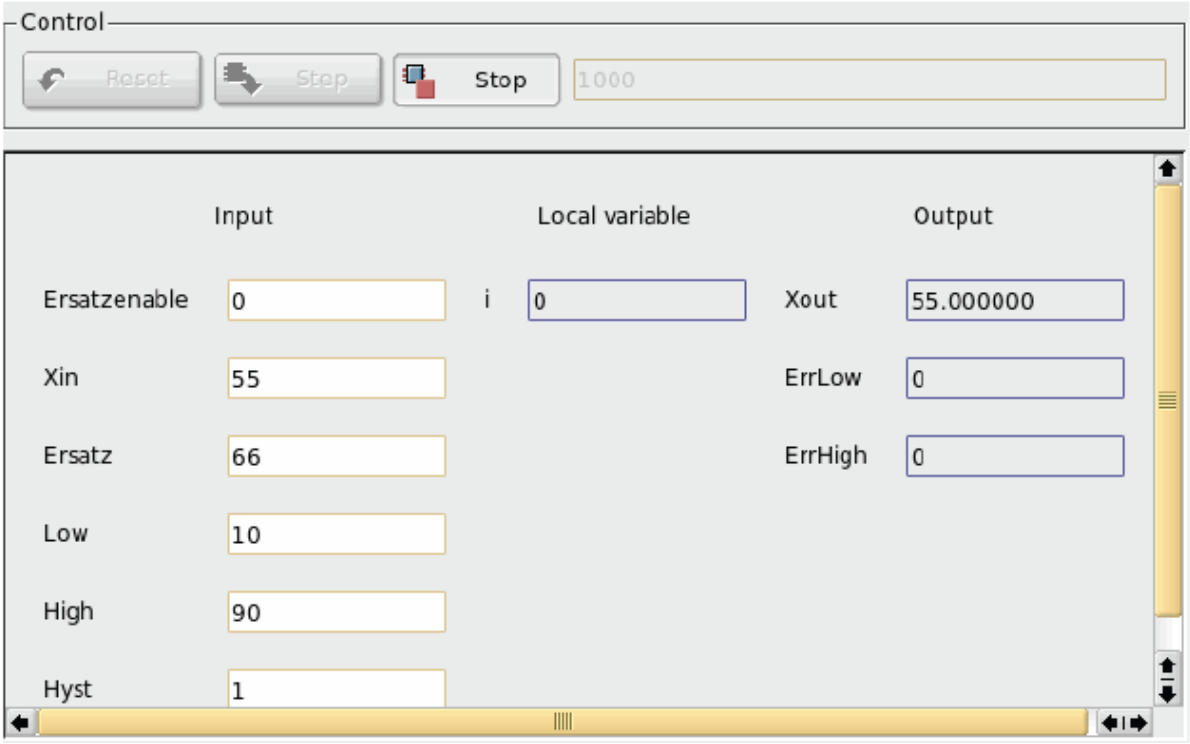


Рис. 19. Отладка функционального блока

3.8 Реализованный функциональный блок

После реализации функции или функционального блока в дальнейшем их можно использовать в гипер-макросах (разработка библиотек) или структурных схемах (разработка проектов). В функции или функциональном блоке может содержаться как 3 строки кода на C, так и более сложная программа с предложениями "include" и "define". Для разработчика практически не существует ограничений.

Рис. 20. Ход разработки

Упражнение 2:

Создать функциональный блок, отслеживающий аналоговое значение. Необходимо установить нижнюю и верхнюю границы, а также гистерезис. Помимо этого необходимо реализовать замещающее значение, на которое можно переключиться при помощи флага включения. Проверить работу функционального блока в библиотеке при помощи отладчика.

4. СОЗДАНИЕ МАКРОСОВ КАРТИНОК

Создание **макросов картинок** и **блоков картинок** уже было рассмотрено в курсе обучения APROLINT. В последующих курсах обучения ***APROL*** будут разъяснены дополнительные возможности и функции. Помимо этого будут затронуты вопросы назначения прав пользовательского управления.

4.1 Создание макросов картинок

Для создания макроса картинки можно использовать пункт меню **Create part / Picture macro (Создать элемент/ Макрос картинки)**. После создания макрос можно открыть и редактировать в редакторе DisplayEditor. В курсе обучения APROLINT было рассмотрено несколько способов создания макросов анимированных рисунков. Для создания действий, например, **выполнения команд** или **ввода значений**, можно использовать элементы **управления** и **визуализации**.

Что такое элемент управления?

Поверхность элемента управления обычно невидима в DisplayCenter. Это **область, реагирующая на нажатие кнопок мыши**, в которой при щелчке выполняются действия. **Возможность управления** отображается при помощи **изменения указателя мыши** в DisplayCenter.

Полезно разместить мини-приложение (например, кнопку или графический символ) над элементом управления для его обозначения.

При запуске DisplayCenter с параметром **"highlightControl" поверхность элемента управления** выделяется **рамкой** при перемещении в нее указателя мыши.

Место расположения элемента управления может размещаться в любой точке диаграммы процесса. Удобно размещать элементы управления сверху или снизу значка управляемого устройства.

Имеются следующие элементы управления:

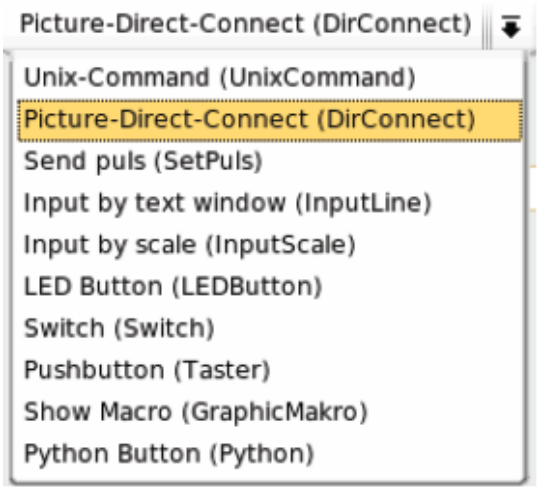


Рис. 21. Имеющиеся элементы управления

Имя элемента управления	Краткое описание
Unix-Command	Данный элемент управления позволяет выполнять команды UNIX на диаграмме процесса. Служит только для совместимости. Замена: Python
Picture-Direct-Connect	Picture-Direct-Connect: данный элемент управления может использоваться только при создании диаграмм процессов. Позволяет выбирать диаграмму процесса внутри другой диаграммы процесса.
Send pulse	Send pulse: подать импульс на цифровую переменную. Можно указать длительность импульса.
Input by text window	Input by text window: открывает окно ввода, в котором можно изменить значение PV.
Input by scale	Input by scale: позволяет изменить значение PV масштабированием.
LED button	Светодиодный индикатор позволяет изменить значение цифровой переменной.
Switch	Переключатель используется для изменения значений цифровой переменной.
Push button	Push button: определяет клавишу для установки логического значения ранее определенного PV.
Show macro	Show macro: При включении этот элемент управления открывает другое окно.
Python button	Кнопка, позволяющая выполнять код на Python в фоновом режиме. Для изучения данной возможности предполагается разработать дополнительный курс обучения (TM870).

Чтобы создать **элемент управления**, используйте пункт меню **Tools** / **Draw** / **Buttons (Controls)) Инструменты/ Рисование/ Кнопки (элементы управления)** в редакторе **DisplayEditor** или нажмите кнопку **Control (элемент управления)** на панели инструментов.

Щелкните кнопкой мыши для определения расположения левого угла **элемента управления**. **Элемент управления** создается в виде прямоугольника и размещается аналогично любым другим функциям рисования. Помимо этого, **элемент управления** можно вытягивать в любом направлении. После придания **элементу управления** требуемого вида щелкните еще раз кнопкой мыши. Прямоугольник отметит область реагирования в диаграмме процесса.

После определения конечной точки элемента управления откроется следующее окно:

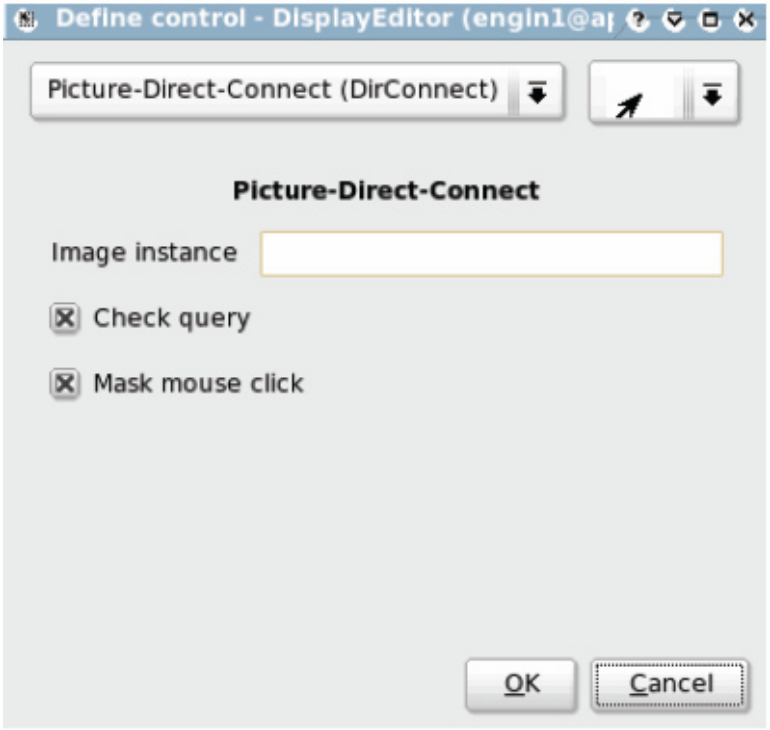


Рис. 22. Создание элемента управления - выбор

Текущий указатель мыши показывается в поле рядом с **кнопкой указателя**.

Сначала используйте **кнопку указателя** для определения указателя мыши, который должен появляться при ее перемещении по элементу управления на диаграмме процесса. Затем откройте **поле выбора**, в котором отображается список возможных элементов управления.

Список выбора позволяет определить **тип элемента управления**, который затем можно индивидуально настроить.

Элементы управления создаются в специально спроектированном слое **элементов управления**. В нем отключены функции меню **Tools** / **Draw (Инструменты/ Рисование)** и некоторые сопутствующие меню.

При выборе параметра **Object list and all layers (Список объектов и все слои)** в раскрывающемся меню можно редактировать, перемещать, удалять, копировать и т.д. элементы управления независимо от текущего слоя.

Также допускается редактирование существующих элементов управления. Для этого сначала включите **Controls layer (слой элементов управления)**. Затем выберите пункт меню **Edit objects** / **Attributes (Редактирование объектов**/ **Атрибуты)** или нажмите соответствующую кнопку. Затем щелкните элемент управления, который требуется изменить. Откроется окно **Define control (определение элемента управления)** с уже настроенными для него параметрами. В окне можно внести изменения обычным способом.

Что такое элемент визуализации (мини-приложение)?

Элемент визуализации или **мини-приложение** - это **заранее определенный графический элемент**, появление или форму которого можно изменить только при помощи создания свойств.

Доступны следующие элементы визуализации

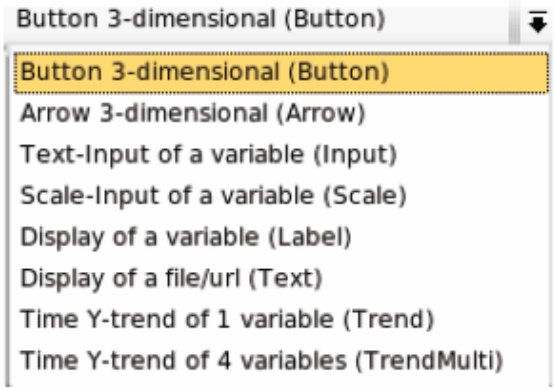


Рис. 23. Список элементов визуализации (мини-приложений)

Имя мини-приложения	Краткое описание мини-приложения
Button 3-dimensional	3-dimensional button: данное мини-приложение показывает 3D-кнопку с текстом.
Arrow 3-dimensional	3-dimensional arrow: мини-приложение создает кнопку в виде 3D-стрелки.
Text input of a variable	Text input of a variable: мини-приложение создает поле ввода, в котором может быть введено значение PV при помощи клавиатуры.
Scale input of a variable	Scale input of a variable: мини-приложение отображает ползунок, с помощью которого можно менять значение PV.
Display of a variable	Display of a variable: мини-приложение отображает значение PV.
Display of a file/url	Display of a file/ URL: указывает содержимое файла. Если указан URL, для отображения соответствующей страницы запускается веб-браузер.
Time Y-trend of 1 variable	Time Y-trend of 1 variable: мини-приложение позволяет отобразить зависимость значения PV от времени в виде кривой.
Time Y-trend of 4 variables	Time Y-trend of 4 variables: мини-приложение позволяет отобразить зависимость значения четырех PV от времени в виде кривой.

Мини-приложения создаются аналогично элементам управления. Однако необходимо помнить, что существует **дополнительный слой Widget (мини-приложений)**, на котором допускается их модификация после создания.

4.2 Готовый макрос картинок

Теперь макрос картинок готов для выполнения действий. Для использования его функциональных возможностей необходимо создать блок картинок, ссылающийся на указанный макрос. При этом создается изображение контактов для связи с гипер-макросами или структурными схемами.

Рис. 24. Еще одна страница (VISU) перевернута

Упражнение 3:

Создать макрос картинок, который подходит к уже созданному функциональному блоку. Это означает, что четыре поверхности ввода и кнопку необходимо подключить к ранее созданному блоку (к контактам low, high, replace, hyst, enable_replace). Помимо этого должно отображаться аналоговое значение (в текстовом виде, в виде гистограммы или обоих) и выделяться цветом замещающее значение.

5. СОЗДАНИЕ БЛОКОВ КАРТИНОК

Блок картинок представляет собой связь между приложением визуализации (динамическими макросами) и логикой.

> **Примечание**
>
> При создании блоков картинок убедитесь, что назначаются макросы картинок, уже находящиеся в определенной библиотеке.

5.1 Создание блоков картинок

Создание **блоков картинок** уже рассматривалось в курсе обучения APROLTR1. В настоящий момент будут создаваться выходы для блока картинок, которые могут представлять права системы управления. Это означает, что в библиотеке могут быть созданы права. Это выполняется, как и при разработке проектов, из меню **Extras** / **Operator rights (Дополнительные возможности/ Права оператора)**. Система позволяет создавать **права с выразительными именами**. Эти права затем назначаются выводам блока картинок.

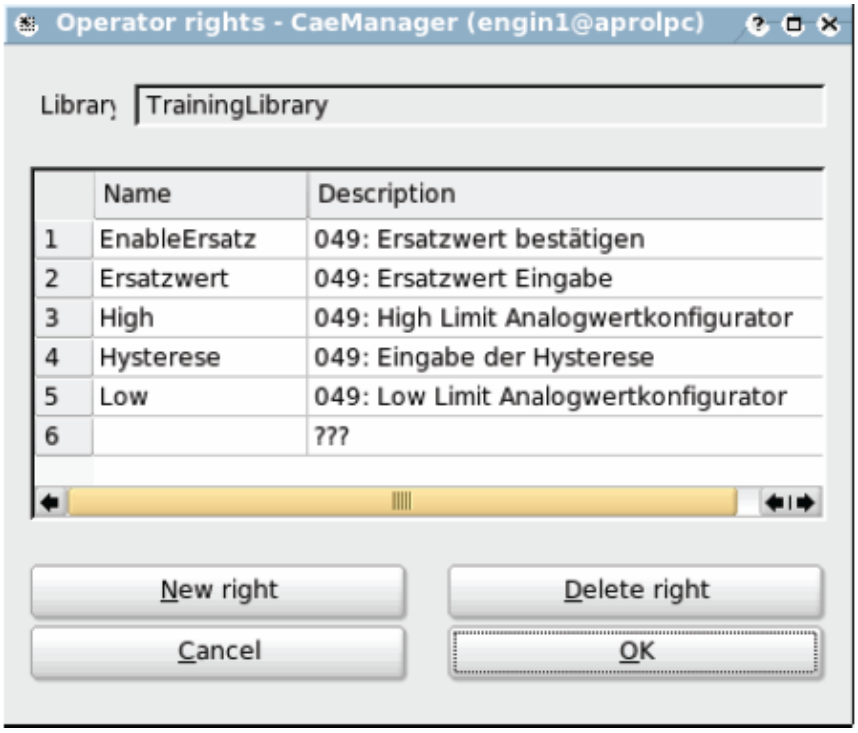


Рис. 25. Права оператора в библиотеке

Новый блок может быть создан аналогично ранее созданной функции (**Create part** / **Block (Создать элемент/ Блок)**), но в данном случае **блок картинок** можно выбрать. Также необходимо выбрать **соответствующий макрос картинки**. Макрос мог бы быть ссылкой на нарисованный объект.

При открытии блока картинок можно настроить переменные динамизации, выделенные красным в представлении I/O (входы/ выходы, тип данных, значение по умолчанию, права оператора). Это проиллюстрировано на следующем рисунке:

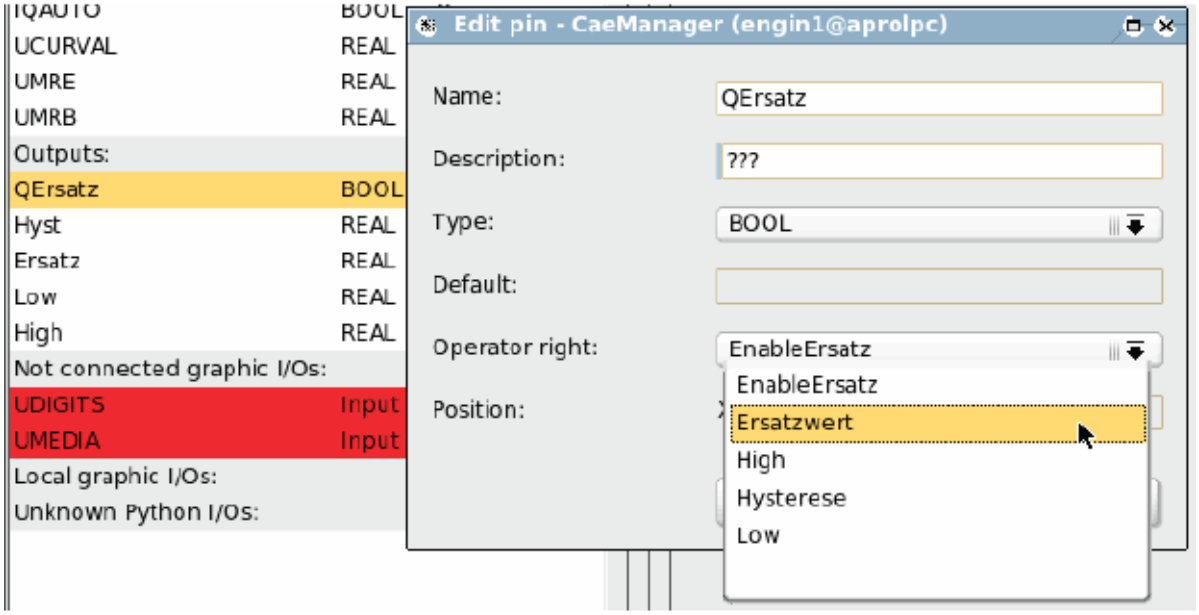


Рис. 26. Редактирование блока картинок

Контакты назначаются посредством выбора записей, появляющихся при щелчке правой кнопкой мыши. При этом также можно редактировать записи.

Подробные сведения всегда можно найти в справочной документации ***APROL***.

5.2 Созданный блок картинок

После успешной компиляции блок картинок готов к использованию. Теперь можно перейти к рассмотрению вопросов анимации.

Упражнение 4:

Создать блок картинок и использовать созданный макрос картинки. Создать логические права и соответствующим образом их назначить.

6. СОЗДАНИЕ ГИПЕР-МАКРОСА

После того, как создан **код на C** и **визуализация**, логично их объединить. Это выполняется при помощи **гипер-макросов**. **Сигнализация и запись трендов** или другие необходимые функции можно объединить в гипер-макрос.

6.1 Общая информация о гипер-макросах

Гипер-макросы значительно упрощают проектно-конструкторские работы за счет повторного использования в структурных схемах уже запрограммированной функциональности. Для решения подобных задач ***APROL*** предоставляет гипер-макросы.

Гипер-макрос позволяет полностью запрограммировать функциональные возможности контроллера, управляющего компьютера, аспекты динамической визуализации, систему сигнализации, систему трендов и т.д. для стандартного приложения.

Элементами гипер-макросов являются блоки, аналогичные блокам структурной схемы.

Разъемы, созданные на входе и выходе схемы гипер-макроса, представляют собой контакты готового блока гипер-макроса, который затем подключается при помощи соответствующих контактов к другим блокам. Помимо этого гипер-макрос содержит контент, который можно использовать для настройки гипер-макросов (текст аварийных сигналов и пр.).

Гипер-макросы обеспечивают следующие преимущества:

- Гипер-макросы можно неоднократно использовать в структурной схеме.
- Использование уже запрограммированной и проверенной функциональности значительно повышает качество проектирования.
- Значительно снижается трудоемкость программирования. Гипер-макросы могут быть вложенными.
- Размещение и структура становятся более понятными.
- Наличие возможности быстрого внесения изменений с обновлением.
- Рост производительности за счет копирования гипер-макросов вместе со свойствами при разработки проектов.

6.2 Создание гипер-макроса

Сначала создайте группу, как уже обсуждалось ранее, в которой будет храниться **гипер-макрос**.

Затем выберите тип **гипер-макроса** в окне **New block (новый блок)**. После этого макрос можно открыть двойным щелчком созданного объекта.

Теперь можно начинать реализацию сложной логики на форме структурной схемы. Программирование **гипер-макроса** выполняется при помощи графического языка функциональных блоков и связей (IEC 1131-3). Затем **гипер-макрос** можно помещать в другие структурные схемы. Отличие от разработки структурной схемы состоит в способе обработки рамки. Создание **разъемов** (щелчком правой кнопки мыши рамки) предоставляет изображение гипер-макроса извне.

Рис. 27. Создание разъема

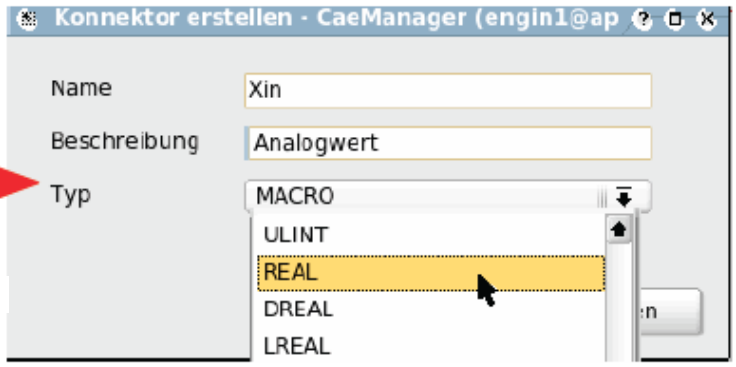


Рис. 28. Настройка разъема

Переменные, которые позднее "соединяются", т.е. подключаются к логическим операциям или PV, должны быть созданы в виде разъемов. Все константы, которые не требуют изменения при работе, могут создаваться как константы гипер-макросов. Это сведет размер гипер-макроса к минимуму. Необходимо учитывать возможность данного способа реализации.

Создавая **константы гипер-макросов** (также щелчком правой кнопки мыши), вы получаете параметры настройки **гипер-макроса**.

Рис. 29. Создание константы HM

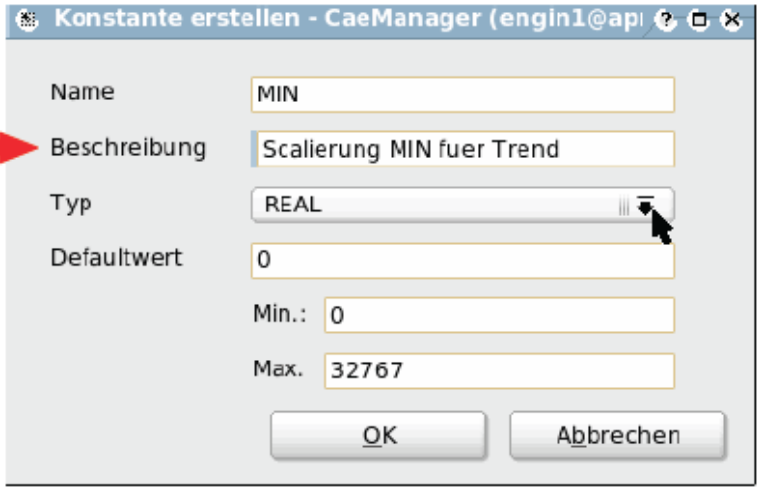


Рис. 30. Настройка константы HM

Типовые константы гипер-макроса включают настройки для систем сигнализации и трендов (например, текста аварийного сигнала, режима сигнализации, группы аварийных сигналов и трендов и т.д.).

При создании константы гипер-макроса ей необходимо указать **значение по умолчанию**. В качестве него может использоваться логическое значение требуемой функции. Кроме этого, необходимо указать минимальное и максимальное значения. Задание минимального и максимального значений логической переменной может предотвратить возникновение ошибок при разработке проектов.

Старайтесь сохранять понятную структуру при разработке **гипер-макросов** (наглядная разработка). Место расположения логики управляющего компьютера, логики контроллера, блоков картинок, систем сигнализации и трендов и т.д. необходимо приблизительно указать до создания **гипер-макросов**. При работе с разъемами удобно использовать структуру шин (параллельные линии) для упрощения "чтения" макроса впоследствии.

При разработке гипер-макроса после определения **блоков картинок**, которые обычно содержат выводы в гипер-макросе, можно включить **систему диспетчерского управления (систему прав)**.

Если создается блок картинок, то в нем содержатся **права по умолчанию**, определенные при его разработке. Щелкнув правой кнопкой объект и выбирая **пункт меню Operator rights (права оператора),** можно изменить **права оператора** определенного блока.

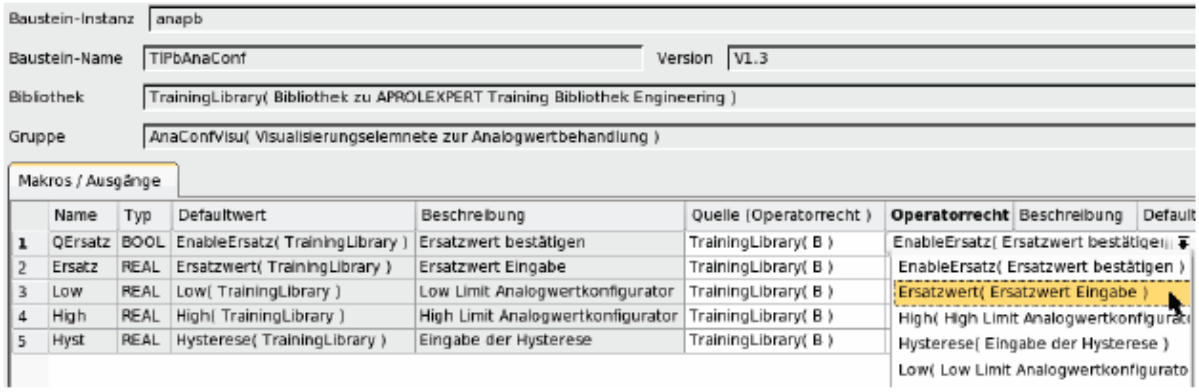


Рис. 31. Изменение прав оператора

Кроме прав оператора, можно ввести значения по умолчанию для положения "всплывающих" окон (в координатах X / Y).

Следующий уровень изменения прав авторизации располагается в приложении разработки проектов (см. введение в ***APROL***).

Всем блокам, используемым в гипер-макросе, необходимо давать по возможности короткие, выразительные имена. Эти имена позднее возникнут при разработке проектов в ходе генерации кода, т.к. на их основе формируются переменные. Существует ограничение на длину имен, связанное с использованием контроллеров B&R, в соответствии с которым длина имени не должна превышать 32 символов. → **Чем короче экземпляр, тем больше символов допускается при разработке проектов.**

6.3 Представление блока гипер-макроса

После создания разъемов гипер-макроса допускается их сортировка для сохранения логического изображения гипер-макроса при проектировании проектов. При этом порядок создания разъемов при разработке гипер-макроса не имеет значения. Разъемы можно сортировать при помощи **представления блока гипер-макроса**.

Представление можно выбрать при помощи пункта меню **View** / **Block (Вид/ Блок)** или поля выбора.

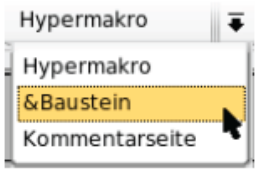


Рис. 32. Выбор представления блока HM

На рисунке показано "стандартное" представление гипер-макроса. Данное представление позволяет перемещать контакты вверх или вниз, щелкая по ним правой кнопкой мыши.

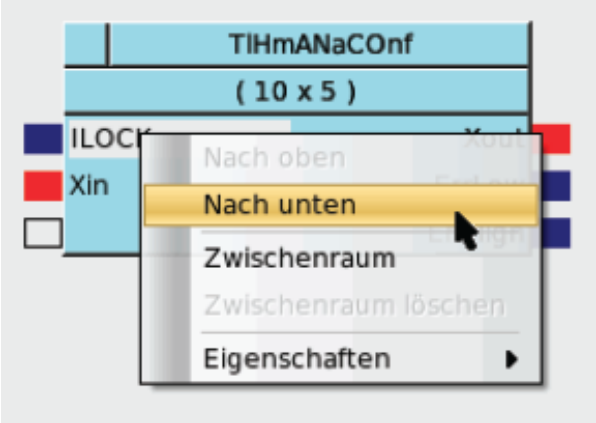


Рис. 33. Перемещение контактов HM

Также это меню можно использовать для резервирования пустого пространства (зазора) для последующего расширения.

Для ссылки на документацию (HTML) по гипер-макросу можно использовать страницу справочной информации или ввести простой текст.

Разъемам гипер-макросов необходимо задать **значения по умолчанию** для гарантии правильной компиляции логики. Для упрощения ввода этих значений необходимо переключиться на **представление значений по умолчанию**.

Рис. 34. Представление значений по умолчанию

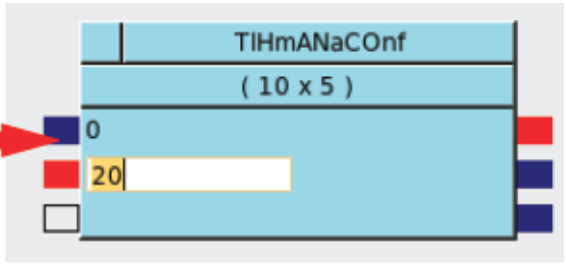


Рис. 35. Ввод значений по умолчанию

Для получения дополнительной информации по этой теме см. справочную документацию ***APROL***.

6.4 Использование гипер-макросов

На данном этапе гипер-макрос полностью реализован. Теперь его можно использовать при разработке библиотек (гипер-макрос внутри гипер-макроса) или при разработке проектов.

Использование гипер-макросов значительно снижает затраты на настройку.

Рис. 36. Сформированная библиотека (при помощи HM)

Упражнение 5:

Создать конфигуратор аналогового значения гипер-макроса с замещающим значением по умолчанию. Реализовать системы сигнализации и трендов в гипер-макросе. Протестировать вновь созданный гипер-макрос в проекте и проверить всю функциональность гипер-макроса в среде ВЫПОЛНЕНИЯ.

7. ПРОЕКТИРОВАНИЕ БЛОКОВ UCB

APROL не утверждает, что предоставляет в своих библиотеках безошибочные функциональные блоки для всех приложений. В ряде случае невозможно реализовать требуемую функциональность при помощи функции или функционального блока. Для этих целей предоставляются блоки USB (блоки пользовательских настроек). Как следует из названия, это функции, которые могут свободно настраиваться пользователем.

7.1 Общая информация

Блоки UCB программируются при помощи скрипта (т.е. исполняемого текстового файла). По этой причине блок UCB может с точки зрения физической реализации использовать только управляющий компьютер. Скрипт позволяет наиболее гибко реагировать на крайне различные требования.

Для программирования может использоваться любой язык скриптов, имеющийся в Linux.

Чтобы не осваивать другие языки скриптов, рекомендуется программировать на Python, поскольку с данным языком вы будете сталкиваться также и в других приложениях.

Блоки UCB выполняются UcbServer. UcbServer - это отдельное приложение ***APROL***, которое всегда доступно в каталоге системы управления процессами для систем СРЕДЫ ВЫПОЛНЕНИЯ. Из данного каталога можно запустить это приложение. Далее описывается работа приложения.

UcbServer определяет конфигурацию всех блоков UCB в системе управления процессами и отслеживает срабатывание на входе.

Если UcbServer обнаруживает передний фронт сигнала на входе "UcbTrigger", вызывается скрипт, относящийся к функции UCB. Таким образом данное событие при помощи скрипта обеспечивает включение любого действия, определенного пользователем.

7.2 Создание блока в библиотеке

7.2.1 Создание блока

Функциональный блок типа **UCB block** включается в библиотеку. И на этот раз действие выполняется при помощи **меню Create part (Создать элемент)**.

7.2.2 Создание параметров и контактов по умолчанию

Каждый блок UCB требует наличия входа **UcbTrigger** с типом BOOL. При поступлении на вход положительного фронта сигнала UcbServer выполняет скрипт.

Помимо этого, UcbServer всегда требует **выход UcbStatus** с типом INT. Сервер записывает на этот выход все ошибки, появляющиеся в процессе выполнения скрипта.

Значение	Описание
0	OK (скрипт присутствует и может быть выполнен)
1	Файл не найден или не заданы права в пути поиска
2	Не найден обыкновенный файл
3	Скрипт не удается выполнить
4	Экземпляр скрипта еще выполняется
5	Объем выходных данных, сформированных скриптом, превышает 1 Мб
6	Экземпляр блока выполняется (включена общая блокировка экземпляра)

Если UcbServer обнаруживает передний фронт сигнала на входе "UcbTrigger", вызывается скрипт, относящийся к функции UCB. Таким образом данное событие при помощи скрипта обеспечивает включение любого действия, определенного пользователем.

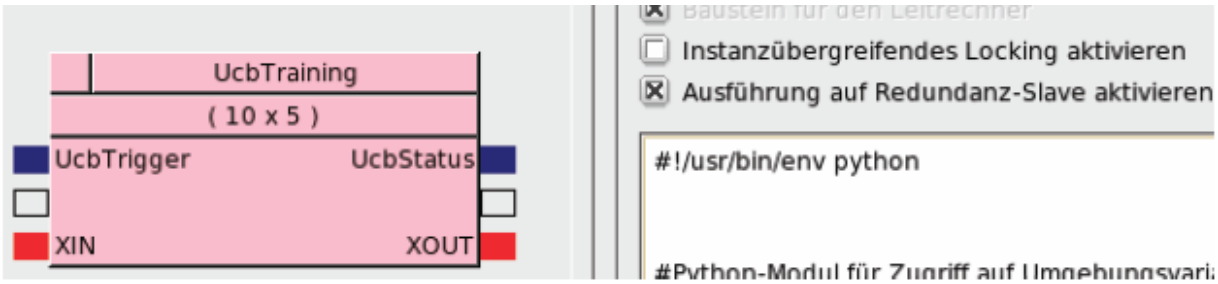


Рис. 37. UcbTrigger, UcbStatus, общая блокировка экземпляра

Для блоков UCB существуют два параметра:

Activate instance-global locking (включение общей блокировки экземпляра):

При выборе данного параметра может одновременно выполняться только один блок. Используемый скрипт определяет, необходим ли данный параметр или нет. При повторном включении блока соответствующий вызов может возвратить на выход UcbStatus код 6 (см. таблицу).

Activate execution on redundancy slave (включить выполнение на резервном подчиненном устройстве):

При установке данного параметра блок выполняется как на главном, так и на подчиненном устройстве при использовании систем с резервированием. И на этот раз используемый скрипт определяет, необходим ли данный параметр или нет.

7.2.3 Создание скрипта

В приведенном примере используется самый простой способ копирования входа блока на его выход. Пример иллюстрирует способ обращения к входным и выходным переменным в скрипте.

В блоке определяется вход XIN и выход XOUT. В дальнейшем переменным XIN и XOUT могут быть назначены значения в логической части.

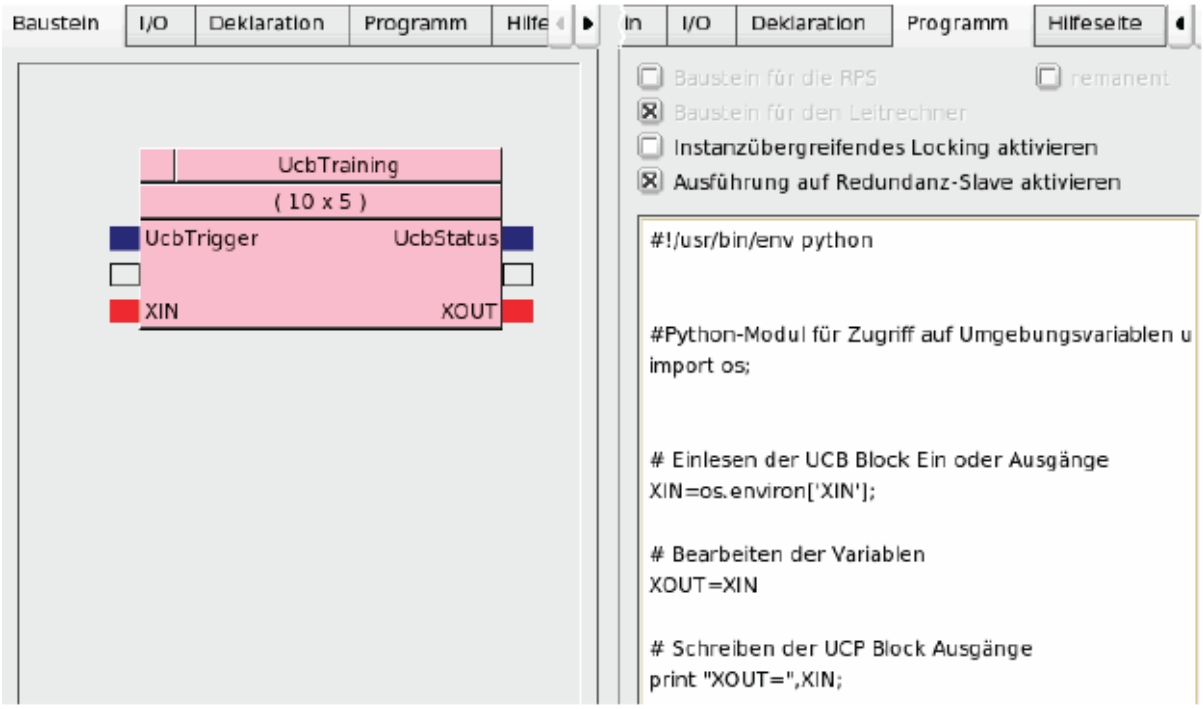


Рис. 38. Пример скрипта UCB

Описание отдельных строк программы в скрипте:

- Задание интерпретатора скрипта

  #/usr/bin/env python

  В первой строке скрипта всегда должен определяться используемый интерпретатор. Обычно интерпретатор - это программа, которая выполняет весь последующий код. В данном случае используется интерпретатор Python.

  При желании может использоваться любой язык скриптов, имеющийся в Linux. В случае использования скриптов оболочки интерпретатор задается в виде #/bin/bash.

- Обращение ко входам и выходам блока

  import os

  XIN=os.environ['XIN']

  К контактам блока можно обращаться следующим образом. Перед запуском UcbServer экспортирует все контакты в так называемые переменные среды. Для обращения к этим переменным используется функция Python os.environ().

  Для этого необходимо импортировать модуль os в скрипт. Вызов os.environ() копирует переменную среды в локальную переменную скрипта. Далее можно использовать эти переменные так, как этого требует логика программы.

- Запись выходных значений

  print "XOUT=",XO A *# For variable values (для переменных)*

  print "XOUT= 123' *# For fixed values (для констант)*

  Вывод значений может быть осуществлен простым указанием переменных в соответствующей команде "Print". Используется следующий формат: VariableName = VariableValue (ИмяПеременной = ЗначениеПеременной). В языке Python используйте команду "print". В скриптах оболочки используйте "echo".

При необходимости существует возможность реагирования в скрипте на запуск или остановку UcbServer. Это может потребоваться, например, при необходимости создать каталог при запуске. Для получения дополнительной информации см. руководство разработчика ***APROL***.

7.3 Тестирование блока в проекте

7.3.1 Настройка сервера UCB

Т.к. UcbServer будет выполнять скрипт, необходимо, чтобы он присутствовал на сервере СРЕДЫ ВЫПОЛНЕНИЯ. Сервер должен быть запущен, в противном случае будут отсутствовать дополнительные параметры для изменения. Это выполняется при разработке модулей для управляющего компьютера.

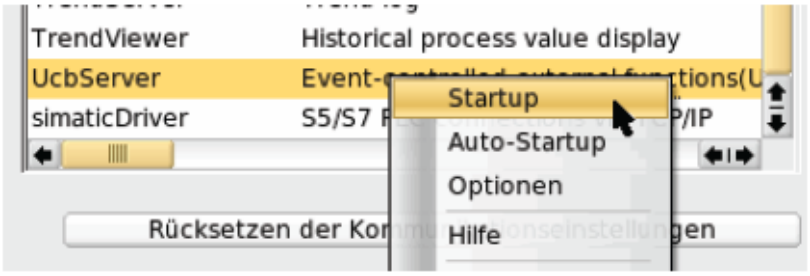


Рис. 39. Проверка запуска UcbServer

7.3.2 Проверка скрипта

После компиляции и загрузки структурной схемы, использующей блок UCB, соответствующий скрипт UCB записывается в каталог $HOME/scripts при запуске UcbServer. В каталоге скрипт ожидает выполнения.

Ошибки при выполнении скрипта регистрируются в журнале в каталоге $HOME/scripts/... в файле <UCB Name>.err.

Установка значения входа UcbTrigger из 0 в 1 приводит к выполнению скрипта. В вышеуказанном примере скрипта входное значение XIN просто копируется на выход XOUT.

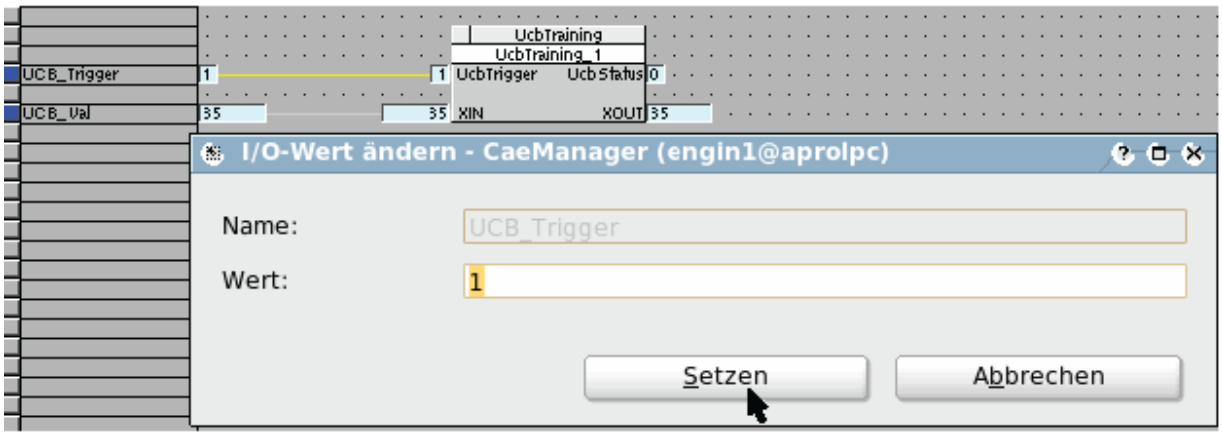


Рис. 40. Тестирование блока UCB при интерактивной отладке

7.4 Тестирование блока без APROL

Для больших по объему скриптов удобно писать фактический код в знакомом вам редакторе (например vi или kwrite) и тестировать их в качестве исполняемого файла. Однако, затем код необходимо скопировать в предназначенную для него область во вкладке **Program (программа)**.

7.4.1 Редактирование скрипта

Скрипт можно выполнить в любом текстовом редакторе. В этом отношении хорошо зарекомендовал себя редактор "kwrite". Его можно запустить непосредственно из меню запуска KDE.

Затем можно сохранить скрипт в любом каталоге, например, $HOME/scripts.

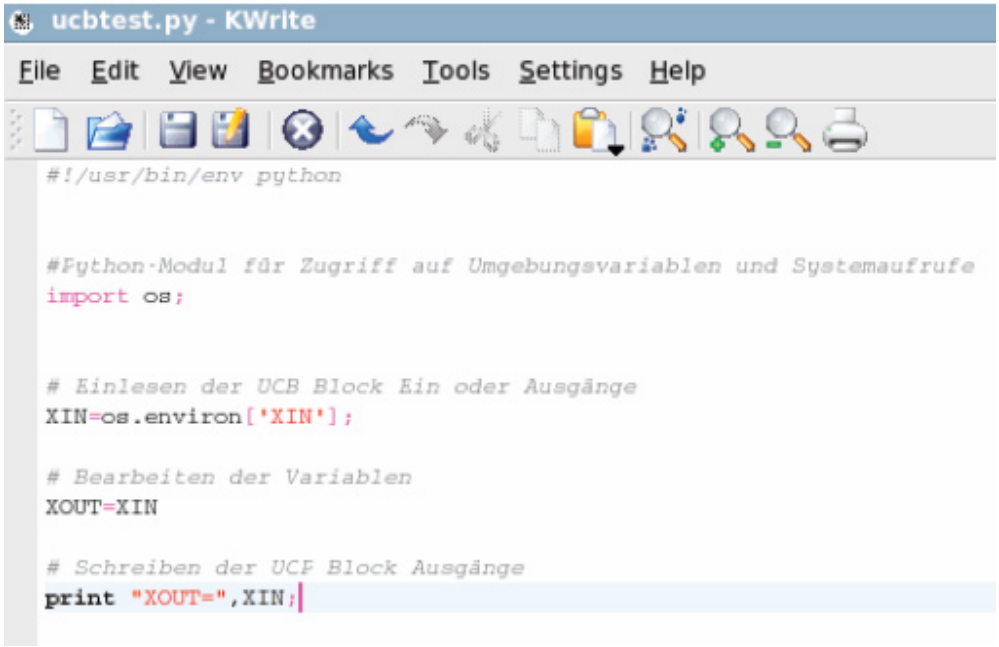


Рис. 41. Скрипт UCB и редактор "kwrite"

7.4.2 Выполнение скрипта

Чтобы выполнить скрипт, запустите консоль ввода. Консоль также может быть запущена из меню запуска KDE. Следующие действия практически совпадают с теми, которые выполняются сервером UcbServer.

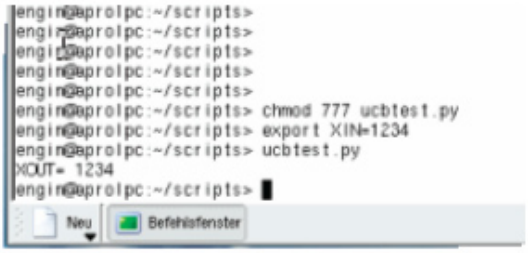


Рис. 42. Выполнение скрипта UCB

Далее необходимо выполнить следующие действия

- Скрипту необходимо задать права на выполнение. Для задания прав можно использовать команду Linux "chmod 777 Scriptname". Это действие необходимо выполнить только в первый раз, т.к. права сохраняются вместе с файлом скриптов.
- Входные контакты необходимо экспортировать в переменные среды. Переменные можно установить при помощи команды "export VariableName = VariableValue (ИмяПеременной = ЗначениеПеременной)". Экспорт должен выполняться каждый раз при перезапуске консоли. Переменная может быть повторно перезаписана при выполнении.
- Скрипт может быть запущен просто исполнением файла со скриптом.

В результате выполнятся все строки скрипта, и все значения, выводимые командой "print", отобразятся на консоли. Выход "XOUT=1234" будет записан UcbServer обратно на выход блока.

7.5 Успешно настроенный скрипт UCB

Программирование скриптов UCB предоставляет различные способы выполнения специфических требований. При использовании языков высокого уровня, таких как Python, ограничения на выполняемые действия практически отсутствуют.

Упражнение 6:

Создать блок UCB, который может считывать имя компьютера, имя пользователя и имя принтера управляющего компьютера. Эта информация содержится в переменных среды HOST, USER и PRINTER. Для обнаружения вызова скрипта необходимо увеличить выходной контакт на единицу при каждом вызове блока.

Упражнение 7:

Создать блок UCB, который выдвигает лоток CD-ROM и извлекает CD.

Примечание:

Для выдвижения лотка привода (извлечения CD) используется команда Linux “eject”.

Чтобы запустить команду в скрипте необходимо использовать команду Python “os.system()” (см. документацию по Python).

В качестве пользователя “root” (суперпользователя) необходимо предоставить права доступа на выполнение “eject” всем пользователям. Это выполняется при помощи команды “chmod 7771 bin I eject”.

8. КРАТКИЕ ВЫВОДЫ

Что было изучено?

Создавая собственную библиотеку по конкретному проекту, мы обеспечили для себя способ планирования функциональности, требующейся в проекте. На данном этапе мы можем настраивать и использовать логику, шаблоны целиком (гипер-макросы) и специальные функции на стороне сервера (UCB).

Для получения дополнительной информации см. справочную документацию ***APROL***.

Рис. 43. На этом разработка библиотек по проекту завершена

9. ПРИЛОЖЕНИЕ

9.1 Список контрольных вопросов (несколько вопросов по разделам)

Что такое функция?

Что такое функциональный блок?

Чем отличается функция от функционального блока?

Понимаете ли вы взаимодействие макроса картинок и блока картинок?

Что такое гипер-макрос? Существуют ли преимущества проектирования при помощи гипер-макросов?

Когда необходимо использовать блок UCB?

9.2 Ответы на контрольные вопросы

Обратите внимание на необходимую дополнительную информацию.

Примечания

Примечания

Обзор учебных программ

TM200 - Презентация компании B&R **
TM201 - Спектр продуктов B&R **
TM210 - Основные сведения об Automation Studio
TM211 - Интерактивное взаимодействие Automation Studio
TM212 - Объекты автоматизации **
TM213 - Среда выполнения Automation Runtime
TM220 - Работа технического специалиста по обслуживанию
TM221 - Компоненты автоматизации и источники ошибок *
TM223 - Система диагностики Automation Studio
TM230 - Разработка структурированного программного обеспечения
TM240 - Многозвенная диаграмма (LAD)
TM243 - Последовательные функциональные схемы (SFC) *
TM245 - Список команд (IL) *
TM246 - Структурированный текст (ST)
TM247 - Основы автоматизации (AB) *
TM248 - ANSIC
TM250 - Управление памятью и хранение данных
TM260 - Библиотеки Automation Studio I

TM400 - Основные сведения об управлении перемещениями
TM402 - Системы управления перемещениями с привязкой к координатам *
TM410 - Основные сведения об ASiM
TM440 - Основные функции ASiM
TM441 - Функции ASiM с несколькими координатными перемещениями
TM445 - Программное обеспечение ACOPOS ACP10
TM450 - Концепция управления и настройка ACOPOS
TM460 - Запуск двигателей *

TM410 - Основные сведения о визуализации
TM410 - Основные сведения об ASiV
TM630 - Руководство по программированию визуализации
TM640 - Система сигнализации ASiV
TM650 - Интернализация ASiV
TM660 - Удаленные устройства ASiV
TM670 - Расширенные возможности ASiV

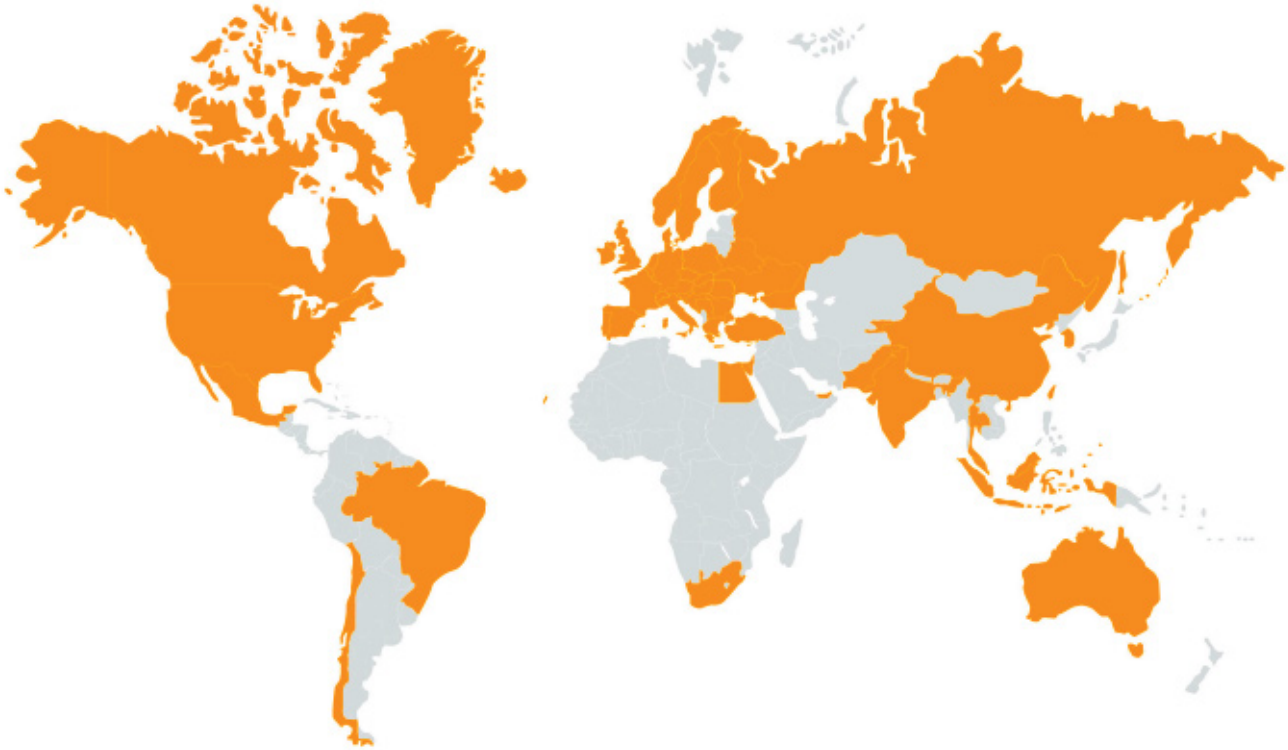
TM700 - Automation Net PVI
TM710 - Средства связи PVI
TM711 - Программирование PVI DLL
TM712 - PVIServices
TM730 - PVI OPC

TM800 - Концепция системы APROL
TM801 - Основы проектирования APROL
TM810 - Установка, настройка и восстановление APROL
TM811 - Система среды выполнения APROL
TM812 - Диспетчерское управление APROL
TM813 - APROL XML-запросы и след контроля
TM830 - Разработка проектов APROL
TM840 - Управление параметрами и наборы команд APROL
TM850 - Настройка контроллера и INA в APROL
TM860 - Разработка библиотек APROL
TM865 - Руководство по библиотекам APROL
TM870 - Программирование на Python в APROL *
TM880 - Отчеты APROL*

*) по запросу
**) см. каталог продуктов

ШТАБ-КВАРТИРА КОРПОРАЦИИ
Bernecker + Rainer Industrie-Elektronik
Ges.m.b.H.
B&R StraKe 1
A-5142 Eggelsberg
Tel.: +43 (0)7748/65 86-0
Fax: +43 (0)77 48/65 86-26
info@br-automation.com
www.br-automation.com

120 офисов более чем в 50 странах - **www.br-automation.com/contact**



Австралия • Австрия • Беларусь • Бельгия • Бразилия • Болгария • Канада • Чили • Китай • Хорватия • Кипр • Чешская Республика • Дания • Египет • ОАЭ • Финляндия • Франция • Германия • Греция • Венгрия • Индия • Индонезия • Ирландия • Израиль • Италия • Корея • Кыргызстан • Малайзия • Мексика • Нидерланды • Норвегия • Пакистан • Польша • Португалия • Румыния • Россия • Сингапур • Словакия • Словения • ЮАР • Испания • Швеция • Швейцария • Тайвань • Таиланд • Турция • Украина • Великобритания • США