2015

СПК

ПЕРВЫЙ СТАРТ

Руководство для начинающих пользователей

Версия: 0.8
Дата: 27.03.2015

Оглавление

1. Введение

1.1. Цель руководства

Целью данного руководства является предоставление пользователю начального количества информации о работе с контроллером серии **СПК**. Руководство затрагивает следующие вопросы:

1. установка необходимого программного обеспечения;
2. настройка связи между контроллером и компьютером;
3. описание интерфейса среды программирования **CODESYS**;
4. подключение к контроллеру модулей ввода-вывода и их конфигурирование;
5. создание и запуск демонстрационного проекта.

Демонстрационный проект, создание которого описано в данном руководстве, ***по умолчанию загружен в контроллер***.

1.2. Общие сведения о СПК

Сенсорный **п**рограммируемый **к**онтроллер (СПК) - это устройство класса **HMI** (человеко-машинный интерфейс), которое совмещает в одном корпусе программируемый логический контроллер с панелью оператора. СПК позволяет не только отображать информацию, но и управлять технологическим процессом в соответствии с заданными алгоритмами, а также архивировать и передавать информацию.

Программирование контроллера (в т.ч. создание пользовательских алгоритмов любого уровня сложности) и разработка экранов визуализации осуществляется в единой среде **CODESYS**, которая распространяется бесплатно и находится на диске с ПО, входящим в комплект поставки.

Руководство по эксплуатации:

контроллеров серии СПК1xx

контроллеров серии СПК2xx

1.3. Общие сведения о среде программирования CODESYS

CODESYS (Controller **De**velopment **Sys**tem) —программный комплекс промышленной автоматизации, основанный на стандарте **IEC (МЭК) 61131-3**. Производится и распространяется компанией **3S-Smart Software Solutions GmbH** (Германия).

CODESYS используется для создания и отладки прикладного программного обеспечения и разработки интерфейса оператора, которые в сочетании образуют пользовательский проект; этот проект загружается на исполнение в контроллер.

Все взаимодействие с контроллером происходит непосредственно с помощью CODESYS, никакое другое программное обеспечение для этого не требуется.

Среда CODESYS находится в процессе постоянного развития и улучшения, что приводит к периодическому выпуску новых версий. Начиная с CODESYS 3.0, версии устанавливаются независимо друг от друга (свежая версия не обновляет предыдущую, а устанавливается параллельно), но при этом ***необходимо*** устанавливать их исключительно в порядке возрастания.

Среда программирования CODESYS полностью **русифицирована**.

1.4. Система версий CODESYS

Среда **CODESYS** находится в процессе постоянного развития и улучшения, что приводит к периодическому выпуску новых версий. Название версии CODESYS выглядит следующим образом:

CODESYS V3.**x** <SP **y**> <Patch **z**> <HotFix **n**>**,** где

V3.x - номер **текущей версии** CODESYS. Новая версия обычно включает в себя принципиальные нововведения, сопровождающиеся серьезными изменениями среды программирования и добавлением значительного количества новых функций;

y - номер **service pack'a**. Сервис пак может вносить изменения, касающиеся интерфейса среды программирования и добавления определенного функционала. Также сервис пак включает в себя все патчи, выпущенные с момента релиза предыдущего сервис пака;

z - номер **patch'a**. Патчи исправляют различные ошибки среды программирования.

n - номер **hotfix'a**. Хотфикс является частной (и крайне редкой) версией патча, оперативно решающей критические проблемы, возникшие после выхода новой версии.

Различные версии CODESYS устанавливаются **независимо друг от друга**; в системе может быть установлено множество версий CODESYS, при этом все они могут быть запущены одновременно, но ***необходимо*** соблюдать указания из п.2.

1.5. Target-файл и прошивка контроллера

Target-файл (файл целевой платформы) является неотъемлемой частью каждого проекта CODESYS. Он содержит информацию о ресурсах контроллера и обеспечивает его связь со средой программирования. Каждая модель контроллера ОВЕН имеет соответствующий target-файл, который необходимо установить перед началом создания проекта в среду CODESYS. Target-файлы входят на диск с ПО из комплекта поставки, а так же доступны в разделе Сервисное ПО соответствующей модели контроллера на сайте owen.ru).

Версия target-файла должна соответствовать версии прошивки контроллера.

Прошивка - это системное программное обеспечение, которое управляет работой контроллера на аппаратном уровне. В связи с добавлением новых функций и исправлением ошибок, регулярно осуществляется выход новых версий прошивок. ***При необходимости*** пользователь может ***самостоятельно сменить*** версию прошивки (вся информация о перепрошивке находится в разделе Сервисное ПО соответствующей модели контроллера на сайте owen.ru).

Версии прошивки и target-файла ***жестко связаны*** между собой; при этом версия CODESYS может превышать версию target-файла, но корректная работа гарантируется только при использовании версий ПО с диска из комплекта поставки.

Подробную информацию о системе версий ПО СПК можно найти в документе «СПК. Система версий ПО», расположенном на сайте owen.ru в разделе **CODESYS V3.x/Документация.**

1.6. Программное обеспечение, использованное в процессе написания руководства

В процессе разработки документа использовался контроллер СПК207.03.CS.WEB с прошивкой 3.944 и ПК со следующим программным обеспечением:

1. ОС Windows 7 SP1 64bit;
2. CODESYS V3.5 SP6 с архивом репозитория V3.5 SP4;
3. Target-файл для СПК207.03.CS.WEB версии 3.5.4.20;
4. Библиотека Util версии 3.5.1.0.

2. Установка ПО

В процессе разработки

В данный момент можно воспользоваться руководством Установка CODESYS V3.5.

3. Первый запуск CODESYS. Создание «пустого» проекта

Запуск среды программирования **CODESYS** осуществляется двойным нажатием **ЛКМ** на соответствующий ярлык на рабочем столе:

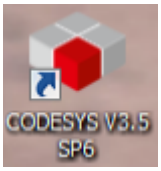


Рис. 3.1. Внешний вид ярлыка CODESYS

При отсутствии ярлыка можно воспользоваться файлом **Codesys.exe**, расположенным в папке ***...\3S CODESYS\CODESYS\Common*.**

В результате откроется **стартовое окно CODESYS**, которое имеет следующий вид:

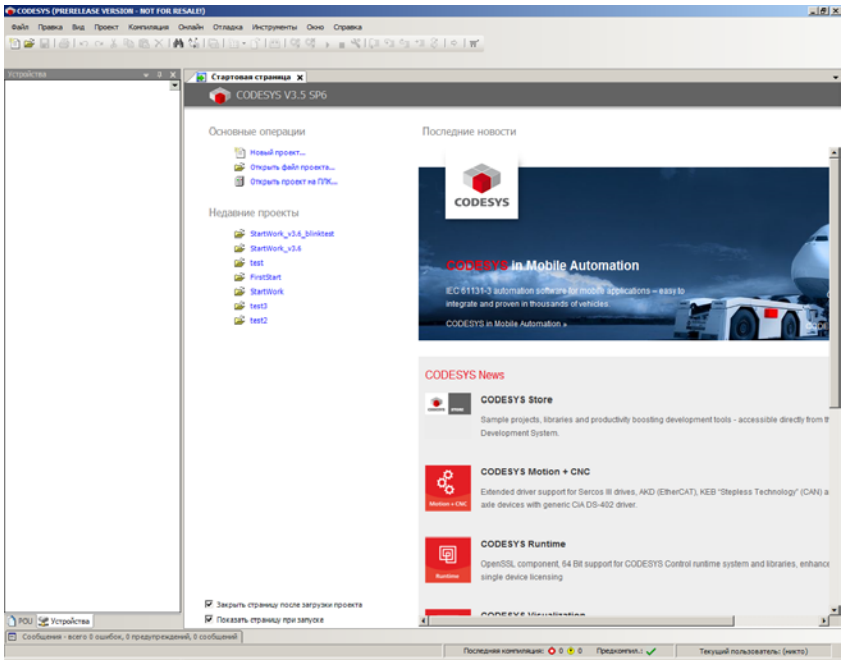


Рис. 3.2. Стартовое окно CODESYS

Для создание нового проекта необходимо нажать на кнопку **Новый проект** (расположена на стартовом экране, а так же находится в меню **Файл**).

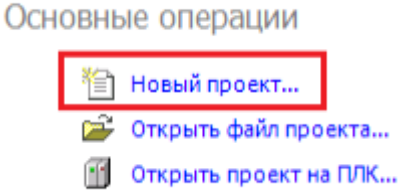


Рис. 3.3. Внешний вид кнопки **Новый проект**

После этого следует выбрать **тип** проекта (или библиотеки, если необходимо создать библиотеку), его **имя** и **папку**, в которой будут сохраняться файлы проекта. Создадим проект типа **Стандартный** с названием **FirstStart**, который будет храниться в папке ***C:\Users\Documents***:

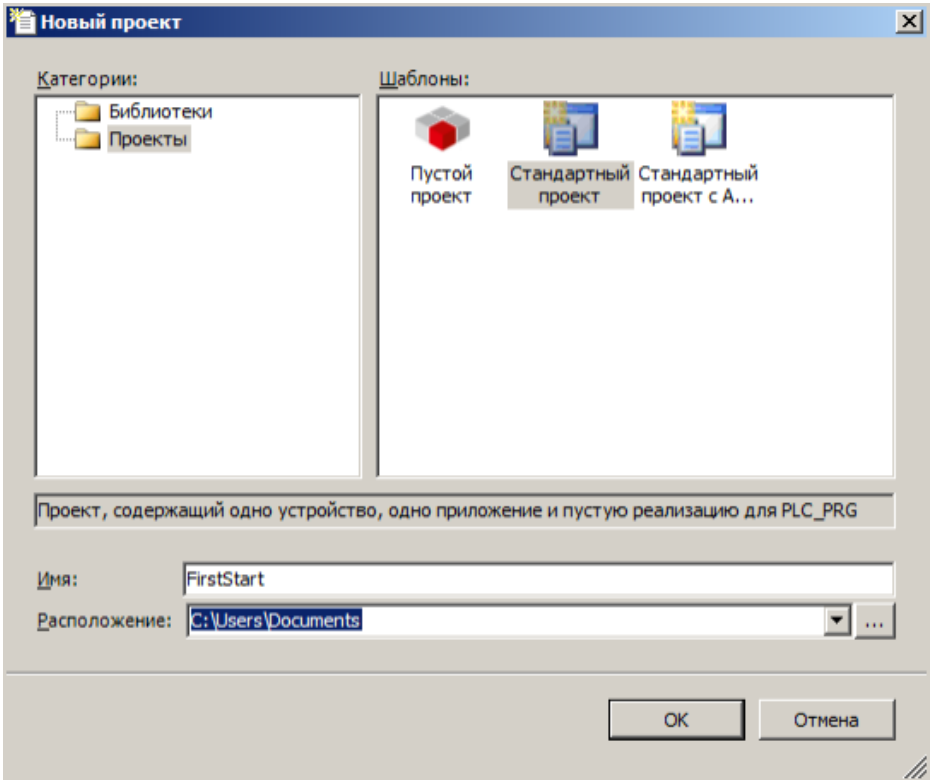


Рис. 3.4. Меню создания нового проекта

В появившемся меню необходимо выбрать **модель устройства** (контроллера) и **язык** создаваемой по умолчанию **программы**. При необходимости модель контроллера и язык программы ***можно будет поменять*** по ходу создания проекта. Выберем устройство (в нашем примере это **СПК207.03.CS.WEB**) и язык **CFC**:

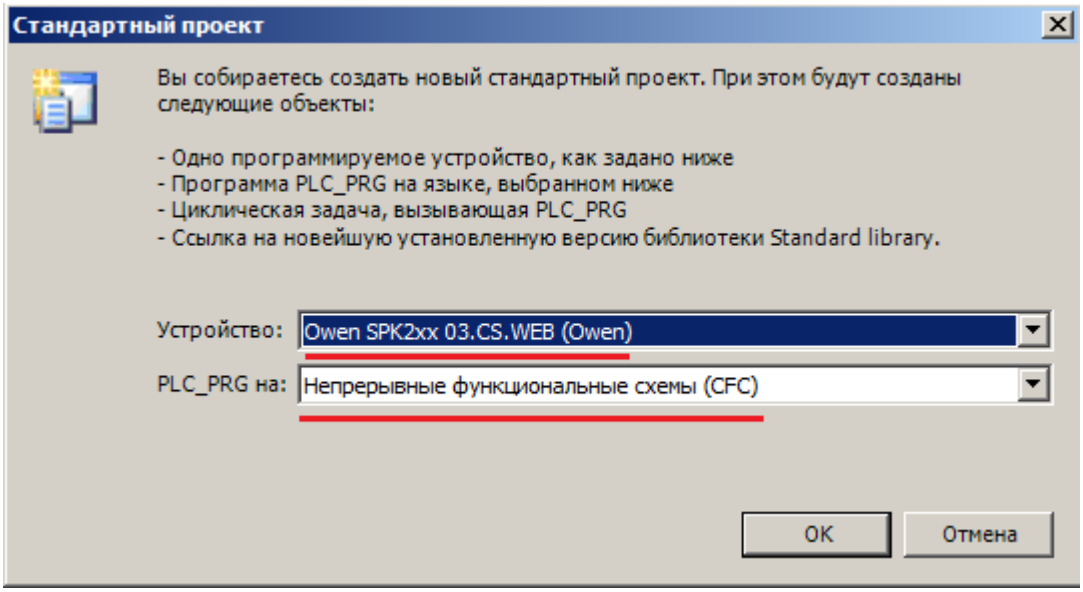


Рис. 3.5. Меню выбора контроллера и языка программирования

При первом запуске **CODESYS** предложит выбрать режим установок среды программирования:

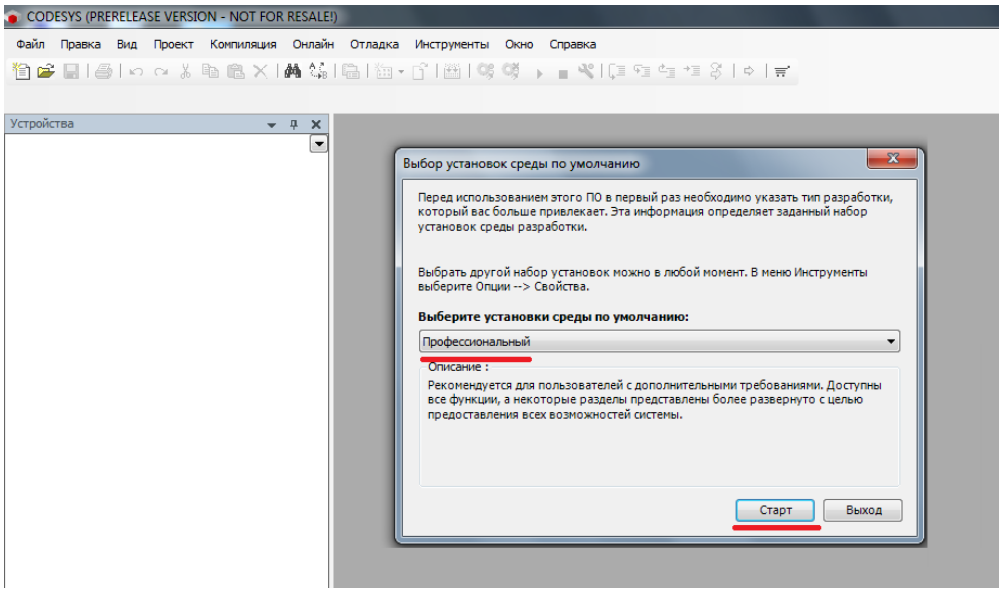


Рис. 3.6. Меню выбора режима установок CODESYS

Рекомендуется использовать **профессиональный** режим, поскольку он предоставляет больше настроек и возможностей по сравнению со **стандартным**. При необходимости режим можно сменить в меню **Инструменты**, вкладка **Опции**, пункт **Свойства**, кнопка **Заданные наборы свойств**:

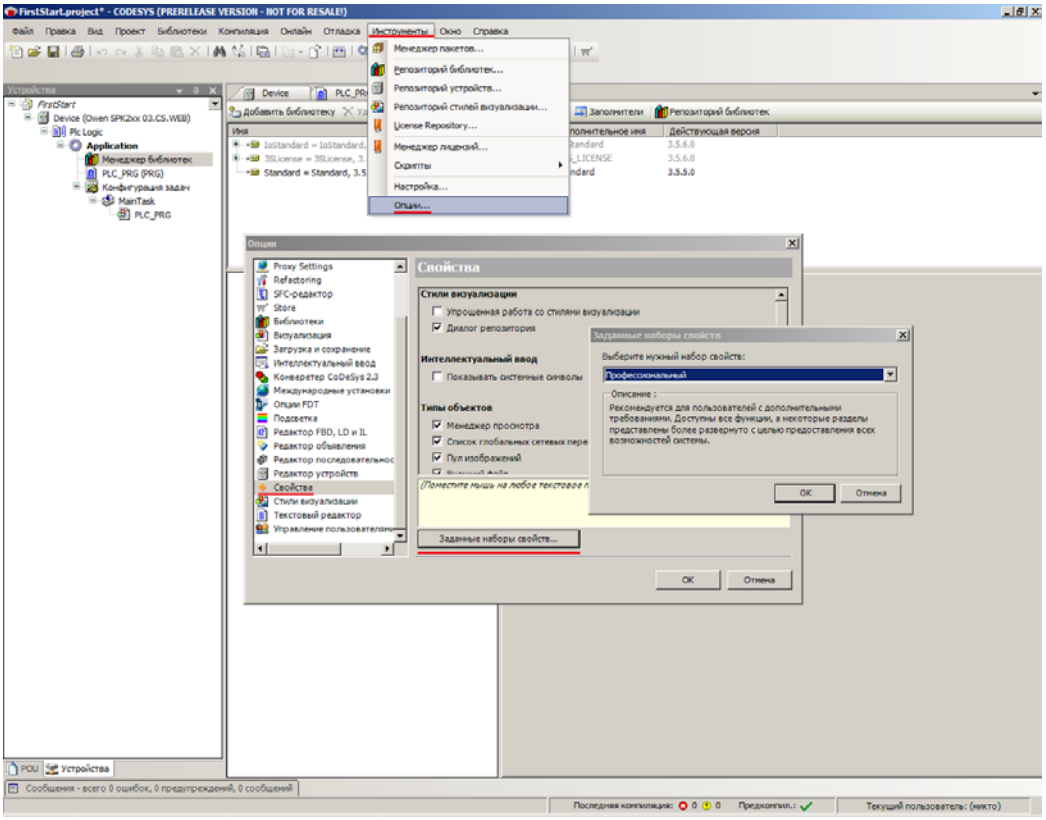


Рис. 3.7. Смена режима установок

По умолчанию среда программирования запускается с **русскоязычным** интерфейсом. При необходимости можно поменять язык в меню **Инструменты**, вкладка **Опции**, пункт **Международные установки**:

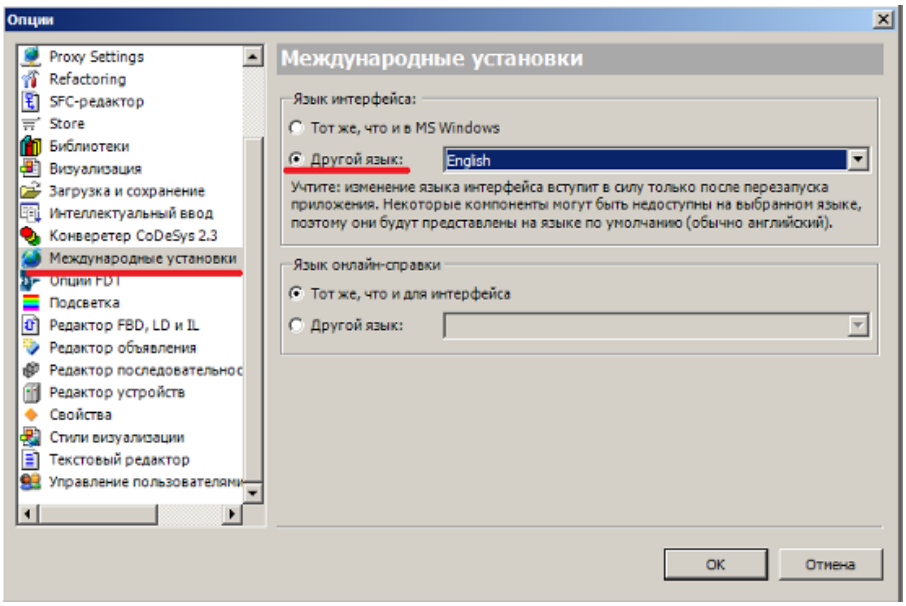


Рис. 3.8. Меню настроек языка CODESYS

Изменения вступят в силу ***после перезапуска*** CODESYS.

4. Интерфейс CODESYS

4.1. Компоненты интерфейса

После создания пустого проекта (см. п.3) окно **CODESYS** будет выглядеть следующим образом:

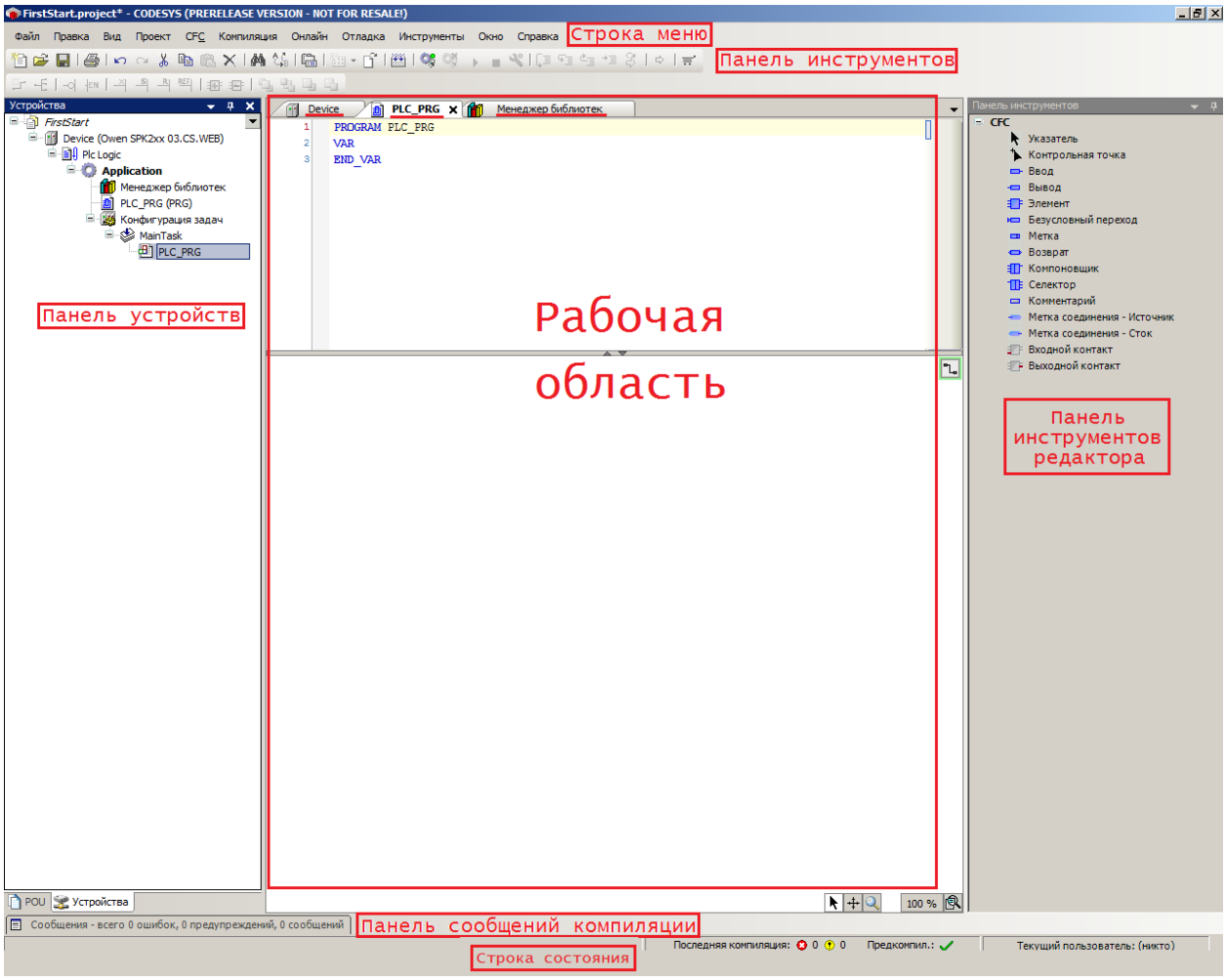


Рис. 4.1. Внешний вид интерфейса CODESYS

На экране расположены следующие компоненты:

1. **Строка меню** - содержит набор меню, используемых при работе над проектом;

2. **Панель инструментов** - содержит набор ярлыков, дублирующих наиболее часто используемые пункты меню;

3. **Панель устройств** - содержит древовидную структуру используемых в проекте компонентов;

4. **Рабочая область** - содержит открытые в данный момент компоненты **Панели устройств**; в зависимости от типа компонентов может использоваться для разработки текстовых и графических программ, создания экранов визуализации, настройки проекта и т.д. Переключение между компонентами осуществляется с помощью **вкладок**, расположенных в верхней части рабочей области;

5. **Панель инструментов редактора (панель элементов)** - содержит функциональные блоки редакторов графических языков программирования и графические примитивы редактора визуализации;

6. **Панель сообщений компиляции** - содержит информацию о процессе компиляции, в частности, об ошибках, возникших в ее ходе;

7. **Строка состояния** - содержит информацию о статусе компиляции, состоянии приложения в режиме **онлайн** (запущено/остановлено), текущем пользователе и т.д.

Подробнее большинство из этих компонентов будут рассмотрены в п. 7, посвященном аспектам создания пользовательского проекта.

Интерфейс **CODESYS** характеризуется значительной гибкостью и может быть настроен как в части количества доступных пользователю меню и панелей, так и в плане количества/ расположения окон.

4.2. Панель меню

Панель меню **CODESYS** при стандартно настроенном интерфейсе имеет следующие пункты:

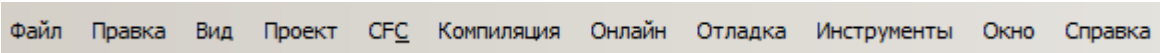


Рис. 4.2. Панель меню

1. **Файл** - содержит команды для работы с файлом проекта (открытие, закрытие, сохранение и т.д.);

2. **Правка** - содержит команды для работы с текстом (копирование, вставка, удаление и т.д.);

3. **Вид** - содержит команды для работы с компонентами интерфейса, в т.ч. команды для открытия/закрытия рабочих панелей. Подробнее данный пункт меню будет рассмотрен в п. 4.3.;

4. **Проект** - содержит команды для работы с компонентами проекта;

5. **CFC** (название может отличаться в зависимости от используемого языка программирования) - содержит команды для работы с редактором соответствующего языка программирования. Данный пункт меню становится доступен только при выделении **POU** (программы) в дереве **Панели устройств**.

6. **Компиляция** - содержит команды, используемые для компиляции проекта;

7. **Онлайн** - содержит команды, используемые для подключения к контроллеру и загрузки в него пользовательского проекта;

8. **Отладка** - содержит команды, используемые при отладке проекта;

9. **Инструменты** - содержит команды, используемые для добавления в проект вспомогательных компонентов (библиотек, target-файлов и т.д.), а также настройки **CODESYS**;

10. **Окно** - содержит команды для управления открытыми окнами и панелями;

11. **Справка** - содержит команды для вызова справки по CODESYS, а также информацию о версии среды программирования.

Меню 1,2,10,11 являются стандартными и не требуют отдельного рассмотрения. Меню 5,6,7,8,9 будут рассмотрены в п. 7, посвященном аспектам разработки пользовательского проекта. Меню 3 и 4 рассмотрены ниже.

4.3. Меню «Вид»

Раскрытое меню **Вид** выглядит следующим образом:

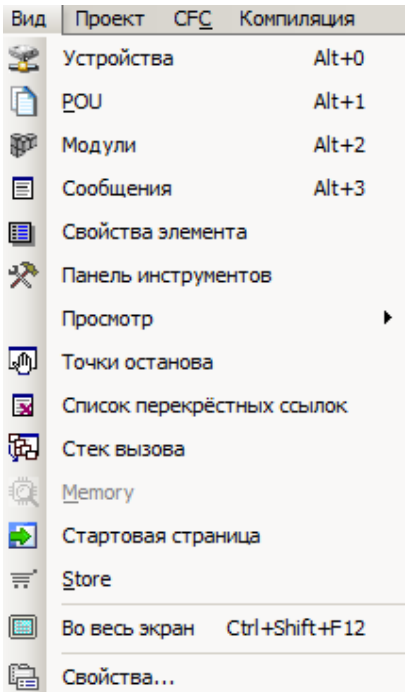


Рис. 4.3. Меню **Вид**

1. Вкладка **Устройства** - открывает **Панель устройств** (см. п.4.1.);

2. Вкладка **POU** - открывает **Панель POU**, содержащую общие для всего проекта компоненты, которые могут использоваться на различных устройствах и в различных приложениях (в отличие от POU из **Панели устройств**, которые строго привязаны к конкретному устройству и приложению).

3. Вкладка **Модули** - открывает **Панель модулей**. Для работы с модулями необходимо расширение **CODESYS Application Composer**, которое **не входит в комплект поставки** и может быть приобретено в **CODESYS Store** (см. ниже);

4. Вкладка **Сообщения** - открывает **Панель сообщений компиляции** (см. п.4.1.);

5. Вкладка **Свойства элемента** - открывает **Панель свойств** для элементов визуализации;

6. Вкладка **Панель инструментов** - открывает **Панель инструментов** (см. п.4.1.);

Вкладки 1-6 используются для настройки интерфейса; для переключения между панелями удобнее будет использовать соответствующие кнопки. Например, для перехода на **Панель устройств** с **Панели POU**:

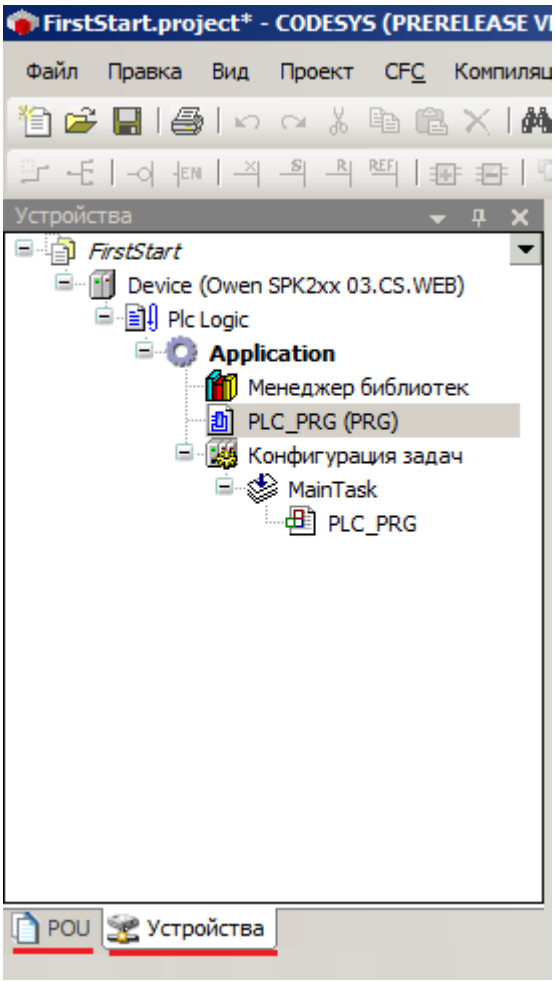


Рис. 4.4. Кнопки переключения между **Панелью устройств** и **Панелью POU**

7. Вкладка **Просмотр** - позволяет создавать списки переменных и просматривать в онлайн-режиме их значения (в процессе работы программы);

8. Вкладка **Точки останова** - позволяет создавать точки останова (используются при отладке проекта);

9. Вкладка **Список перекрестных ссылок** - с помощью этой команды можно открыть окно, содержащее список перекрестных ссылок переменной проекта, т.е. компонентов проекта, в которых используется данная переменная;

10. Вкладка **Стек вызова** - позволяет контролировать процесс выполнения программ в онлайн-режиме;

11. Вкладка **Memory** - позволяет в онлайн-режиме просматривать и сохранять область памяти приложения;

12. Вкладка **Стартовая страница** - открывает стартовую страницу **CODESYS**;

13. Вкладка **Store** - открывает **CODESYS Store** (интернет-магазин расширений среды **CODESYS**);

14. Вкладка **Во весь экран** - разворачивает окно **CODESYS** во весь экран;

15. Вкладка **Свойства** - открывает меню **Свойства** для выделенного компонента **Панели устройств**.

4.4. Меню «Проект»

Раскрытое меню **Проект** выглядит следующим образом:

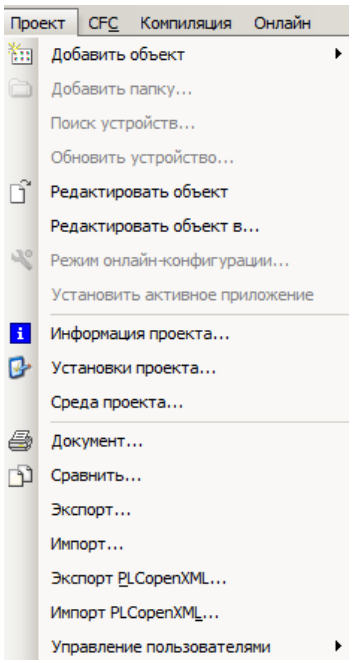


Рис. 4.5. Меню **Проект**

Вкладки **Добавить объект**, **Редактировать объект** и т.д. соответствуют вкладкам из контекстного меню компонентов **Панели устройств**:

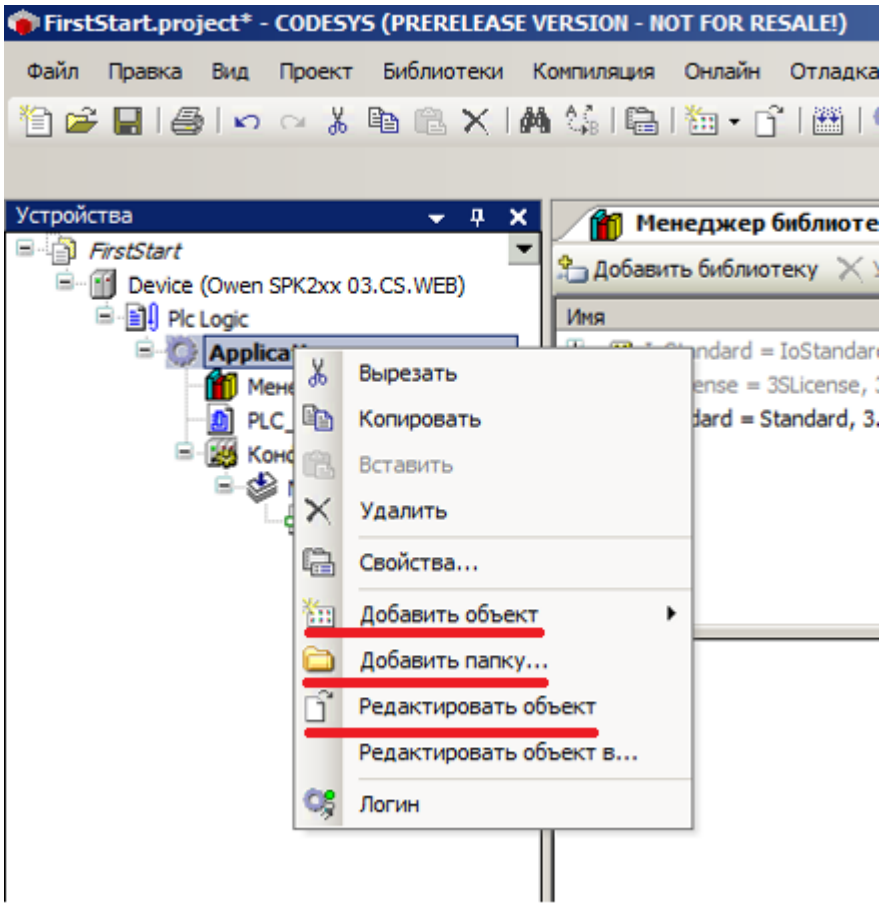


Рис. 4.6. Контекстное меню приложения **Application**

Остановимся на двух наиболее часто используемых вкладках меню Проект - **Информация проекта** и **Установки проекта** (они также присутствуют на **панели POU**):

1. Вкладка **Информация проекта** содержит данные о дате и времени создания проекта, пути его хранения, компании-разработчике, версии проекта и т.д.

2. Вкладка **Установки проекта** содержит общие настройки проекта, касающиеся особенностей компиляции, визуализации, разграничения прав пользователей и т.д.

5. Настройка связи между контроллером и ПК

Различные модификации СПК используют разные интерфейсы для связи с пользовательским компьютером: для СПК2xx – Ethernet, для СПК1xx – USB. Ниже мы подробно рассмотрим процесс подключения моделей СПК2xx (на примере СПК207), а также опишем особенности подключения СПК107/110 и СПК105.

5.1. Настройка связи между СПК2xx и ПК

5.1.1. Сетевые настройки контроллера

Связь между контроллером СПК2xx и пользовательским ПК осуществляется по **Ethernet**. При этом существует **два варианта** взаимного сетевого расположения контроллера и компьютера:

1. Контроллер и компьютер находятся в **одной локальной сети**;
2. Контроллер и компьютер находятся в **разных локальных сетях**, связанных с помощью соответствующих сетевых устройств (маршрутизаторов).

Не имея возможности вдаваться в подробности сетевых технологий (информацию о них легко найти в Интернете), рассмотрим частный (и самый простой) случай **первого варианта подключения** - при котором ethernet-порты контроллера и компьютера соединяются напрямую с помощью кросс-кабеля (**не входит в комплект поставки**). ***Необходимо помнить***, что в локальных сетях не должно возникать конфликта IP-адресов, т.е. разные устройства не должны обладать совпадающими адресами.

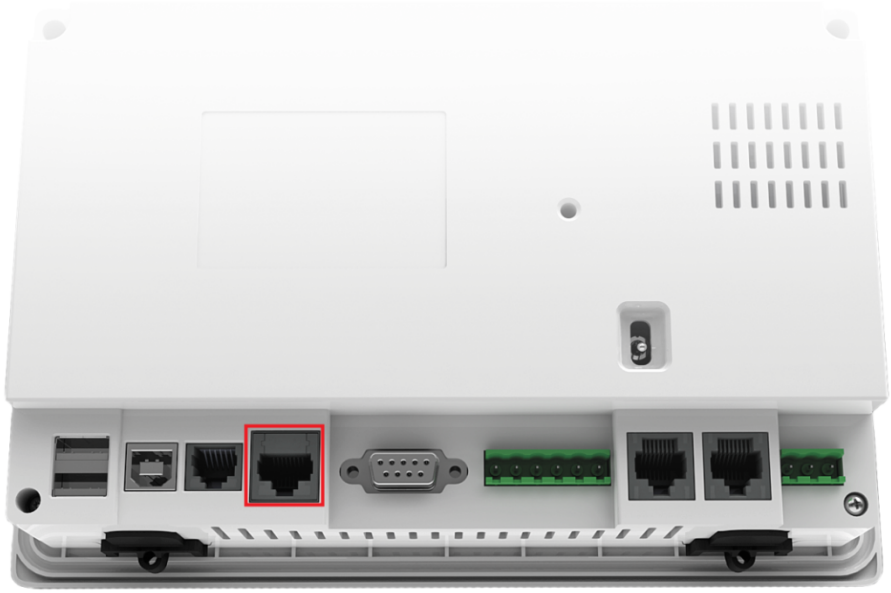


Рис. 5.1. Внешний вид задней панели СПК207 (ethernet-порт выделен красным)

Рис. 5.2. Внешний вид кросс-кабеля

В первую очередь необходимо настроить **сетевые параметры** контроллера.

По умолчанию сетевые настройки контроллера следующие:

режим DHCP – **отключен**

IP-адрес: **10.0.6.10**

Маска подсети: **255.255.0.0**

IP-адрес шлюза: **10.0.6.1**

Имя устройства: **spk207web**

IP-адрес предпочитаемого DNS-сервера: **10.0.6.1**

IP-адрес альтернативного DNS-сервера: **8.8.8.8**

В качестве примера будет рассмотрен процесс подключения к компьютеру контроллера с сетевыми настройками по умолчанию.

Для того, чтобы изменить сетевые настройки контроллера, необходимо после включения питания ***до начала загрузки проекта*** контроллера открыть **сервисное меню**, коснувшись экрана три раза:

Рис. 5.3. Процесс включения контроллера

Для запуска **программы-конфигуратора** следует выбрать соответствующий пункт меню:

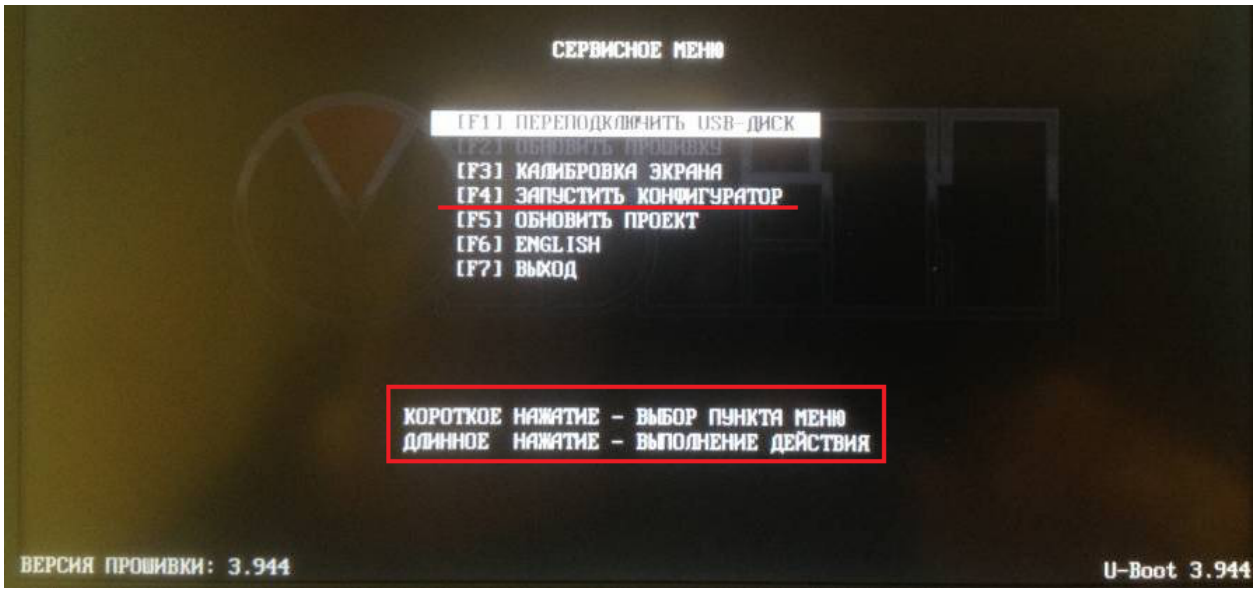


Рис. 5.4. Сервисное меню контроллера

В результате появится стартовое окно программы-конфигуратора. После нажатия на кнопку **Введите пароль** откроется соответствующее диалоговое окно с экранной клавиатурой.

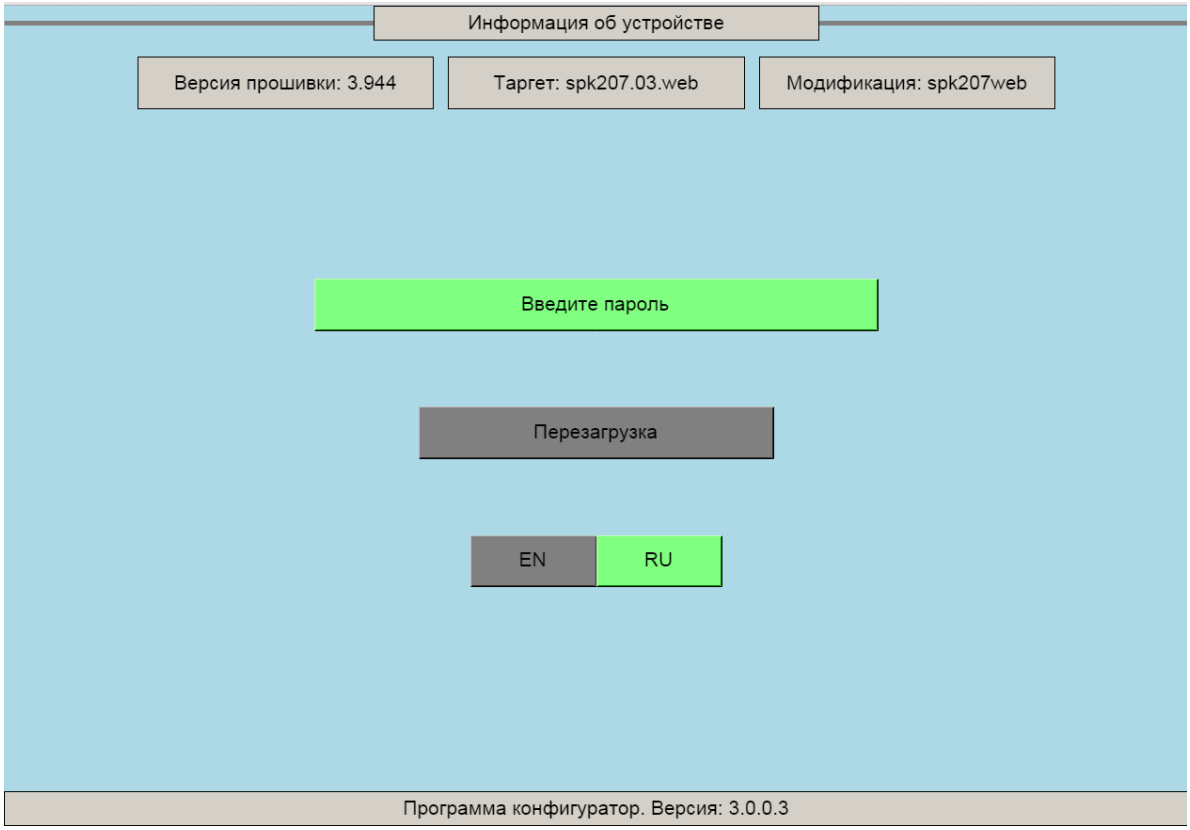


Рис. 5.5. Стартовое окно программы-конфигуратора

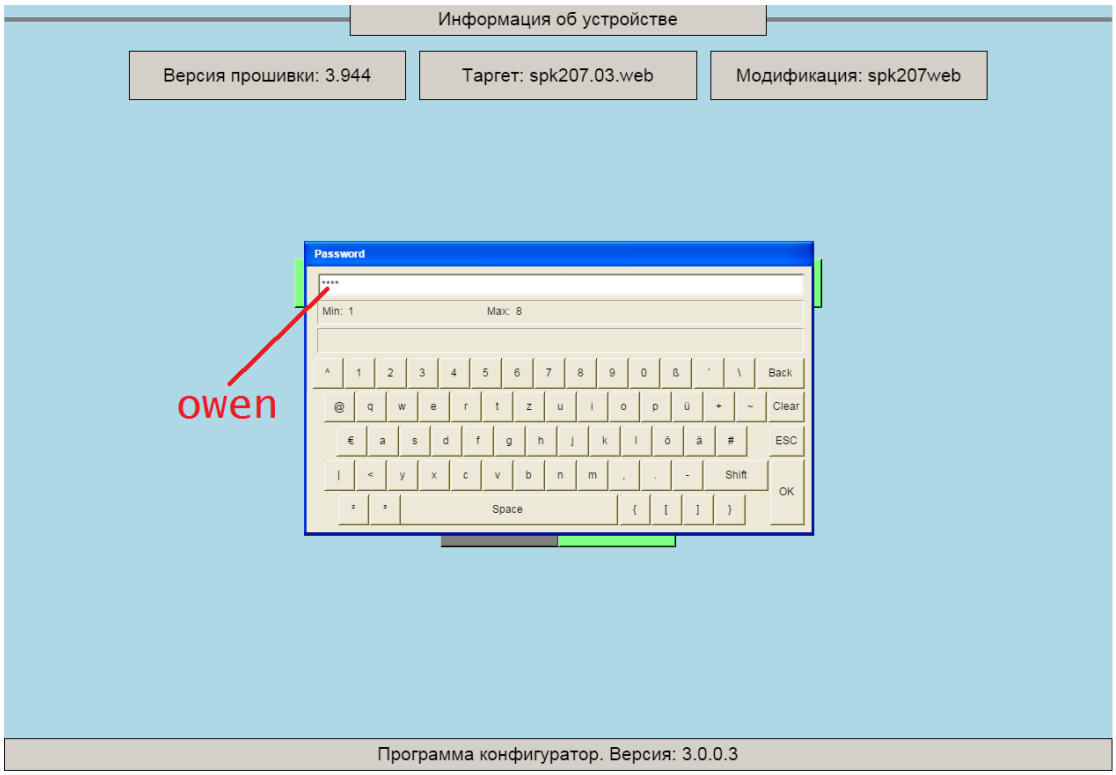


Рис. 5.6. Диалоговое окно ввода пароля

Пароль — **owen**, для переключения экранной клавиатуры на нижний регистр необходимо нажать клавишу **shift**. После ввода пароля следует нажать клавишу **OK**.

Откроется окно конфигуратора, которое имеет следующий вид:

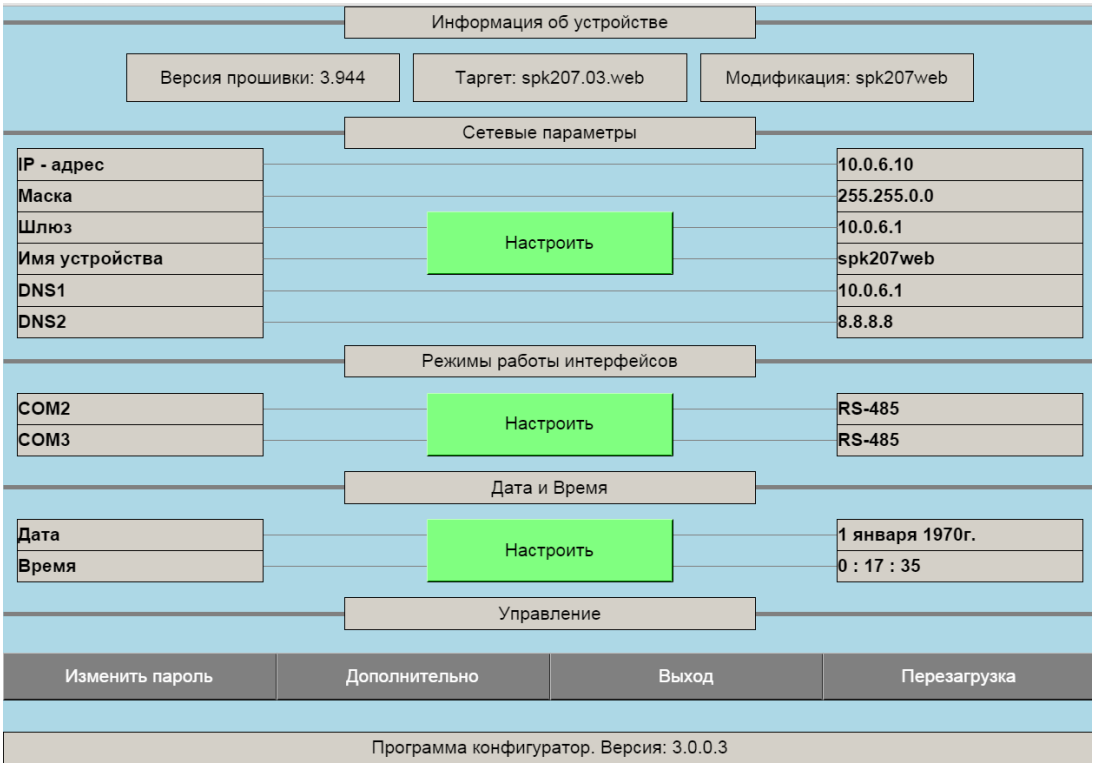


Рис. 5.7. Внешний вид конфигуратора

Конфигуратор предназначен для:

1. Настройки сетевых параметров контроллера;
2. Настройки режима работы последовательных портов (RS-232 или RS-485 соответственно);
3. Установки даты/времени часов контроллера;
4. Настройки ориентации дисплея контроллера, режима работы курсора, отображения информации о состоянии памяти контроллера и подключенных запоминающих устройств (вкладка **Дополнительно**).

Подробное описание конфигуратора приводится в Руководстве по эксплуатации.

Для настройки сетевых параметров необходимо нажать кнопку **Настроить** под соответствующим заголовком.

Рис. 5.8. Меню настройки сетевых параметров

Изменение IP-адресов производится ***нажатием на каждую цифру***, что приводит в появлению экранной клавиатуры. Нажатие на кнопку **DHCP** включает и отключает **режим DHCP** (зеленый цвет кнопки соответствует включенному состоянию, серый — отключенному). Режим DHCP позволяет получить все сетевые настройки ***автоматически*** при наличии в пользовательской сети **DHCP-сервера**. В противном случае настройки нужно менять вручную.

Нажатие на кнопку **Сброс** возвращает сетевые настройки к их значениям **по умолчанию**.

После завершения работы с конфигуратором необходимо перезагрузить контроллер с помощью нажатия на кнопку **Перезагрузка** в конфигураторе, ***не заходя на этот раз в сервисное меню***. В результате после загрузки контроллера запустится демонстрационный проект.

5.1.2. Сетевые настройки компьютера

Рассмотрим процесс настройки сетевых параметров компьютера на примере ПК с операционной системой **Windows 7**.

Откроем **Центр управления сетями и общим доступом** (**Пуск — Панель управления — Центр управления сетями и общим доступом**) и выберем вкладку **Изменение параметров адаптера**.

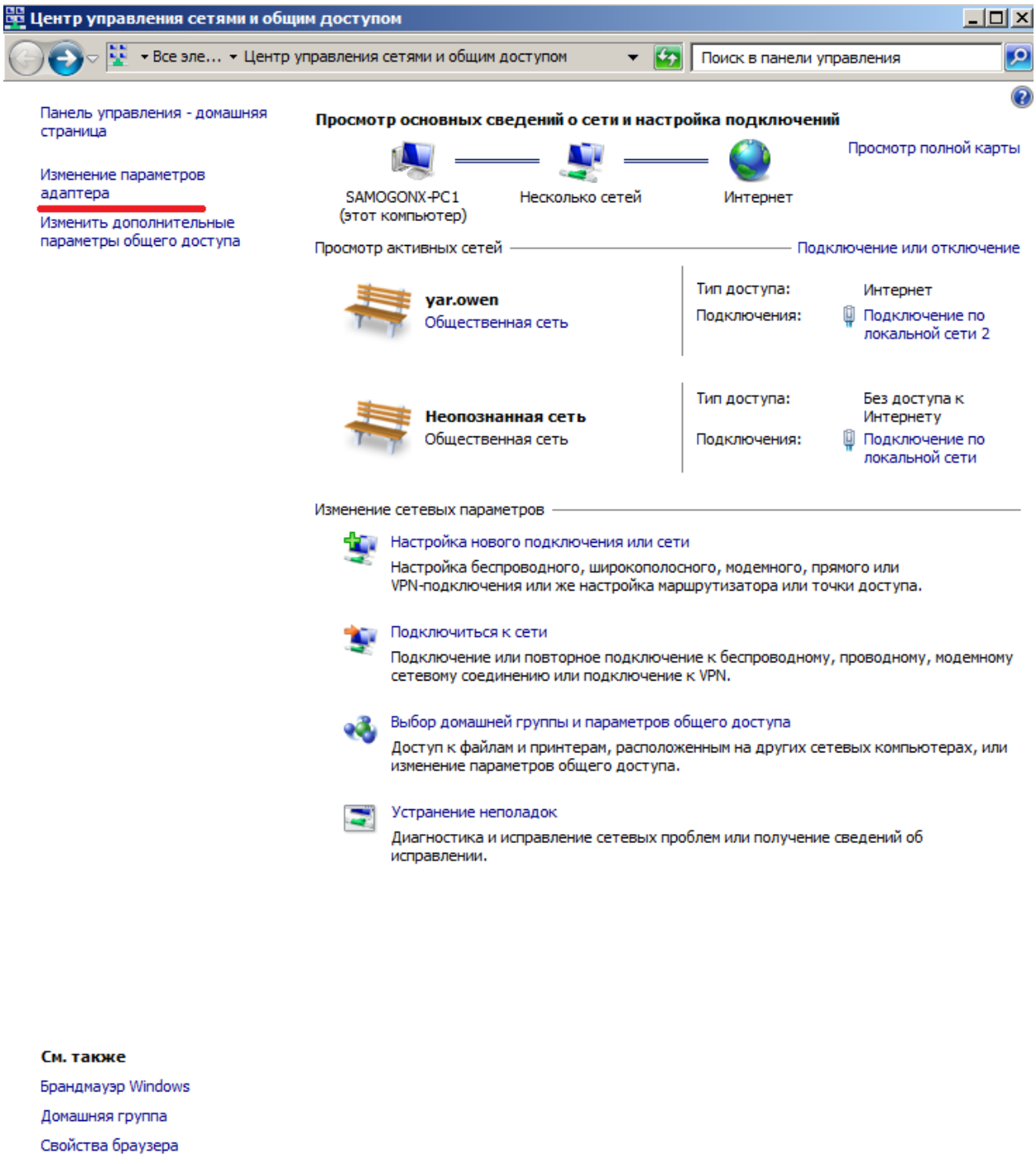


Рис. 5.9. Окно **Центра управления сетями и общим доступом**

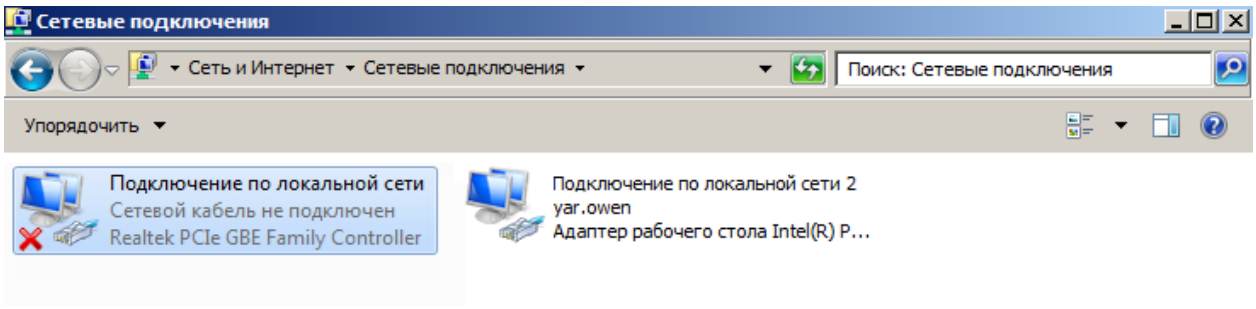


Рис. 5.10. Вкладка **Изменение параметров адаптера** (Сетевые подключения)

Подсоединим один конец кросс-кабеля к ethernet-порту **включенного** контроллера, а второй - к аналогичному порту компьютера. В этот момент один из сетевых адаптеров станет «активным» (с его пиктограммы пропадет красный крест).

Нажмем на нужный адаптер **ПКМ** и откроем вкладку **Свойства**, выберем компонент **Протокол Интернета версии 4 (TCP/IPv4)** и откроем его **Свойства**:

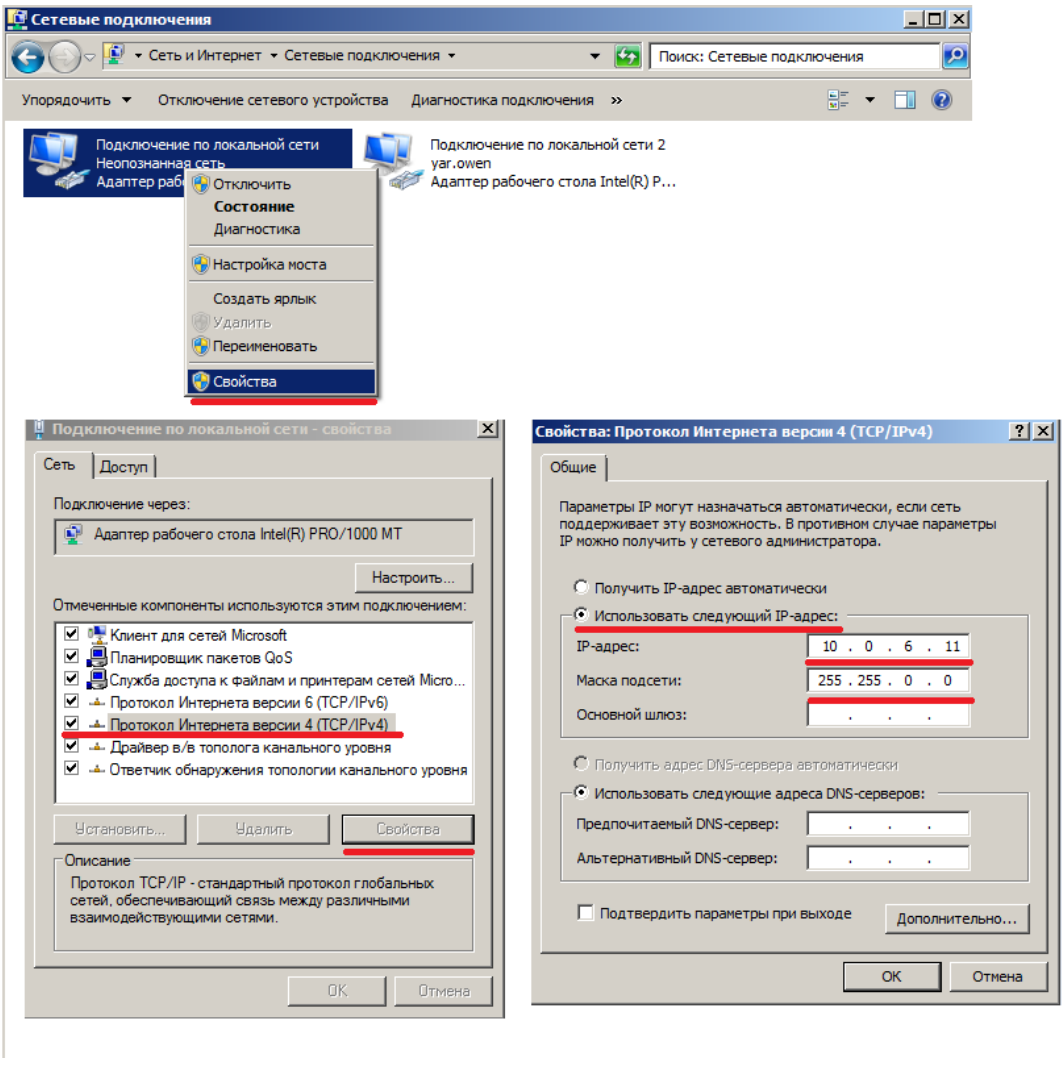


Рис. 5.11. Сетевые настройки компьютера

Зададим компьютеру IP-адрес **10.0.6.11** и маску **255.255.0.0** (***обратите внимание***: IP-адрес **отличается** от IP-адреса контроллера, маска подсети **совпадает** с маской контроллера). Поля остальных настроек оставим пустыми. Нажмем **ОК**. Адаптеру потребуется несколько секунд, чтобы применить новые настройки.

Теперь мы можем проверить наличие связи между компьютером и контроллером. Откроем **командную строку** (**Пуск — Все программы — Стандартные — Командная строка**) и введем команду **ping 10.0.6.10 -t**. При наличии связи мы увидим следующий ответ:

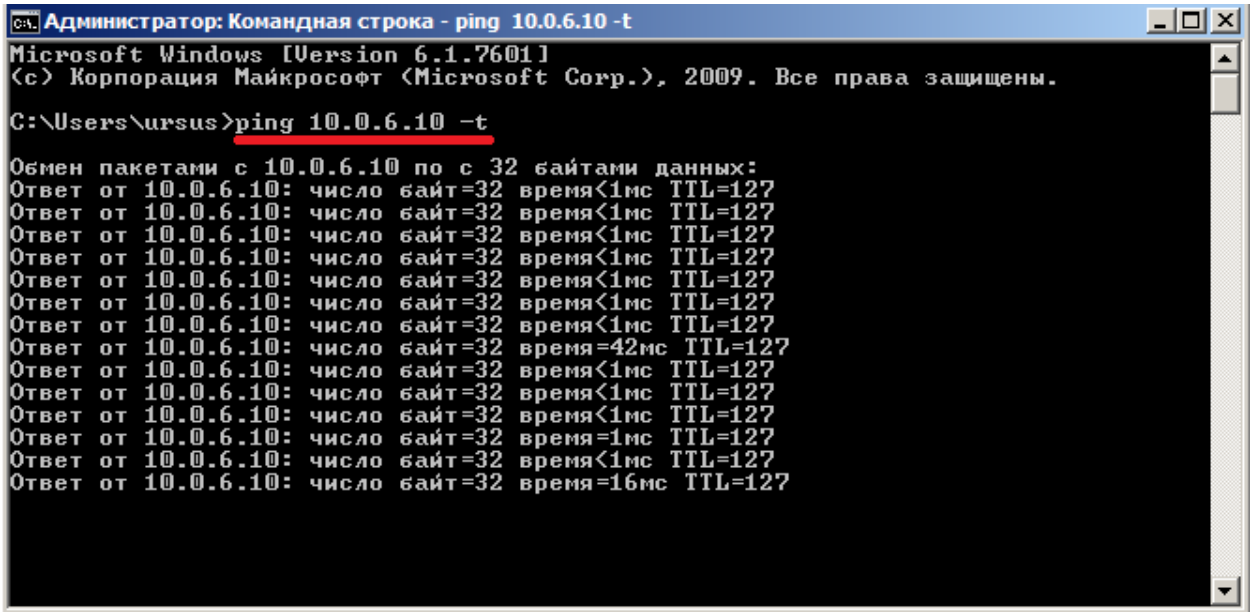


Рис. 5.12. Результат выполнения команды ping

5.2. Особенности настройки связи между СПК107/110 и ПК

Связь между контроллером СПК107/110 и пользовательским ПК осуществляется по интерфейсу **USB** с использованием кабеля типа **A – B**, входящего в комплект поставки:

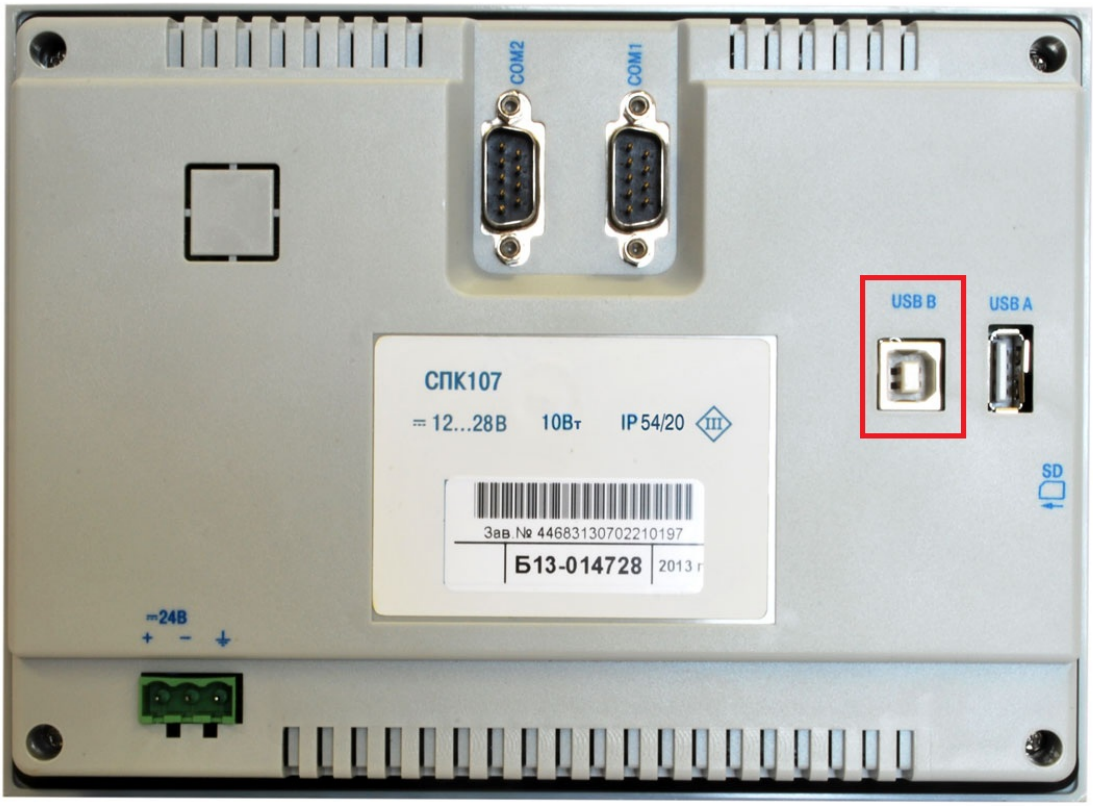


Рис. 5.13. Внешний вид задней панели СПК107 (порт USB B обведен красным)

Рис. 5.14. Внешний вид кабеля USB A – B

Настройка сетевых параметров контроллера ***совершенно аналогична*** настройке контроллеров серии СПК2xx и может быть выполнена по п. 5.1.1. Рассмотрим ***сетевые настройки компьютера***.

При первом подключении СПК1xx к ПК необходимо выполнить установку драйвера USB. Для этого необходимо запустить автоматический установщик, расположенный на диске с ПО из комплекта поставки - **SPK_USB_Driver_V.1.5.102**. При этом может возникнуть следующее сообщение:

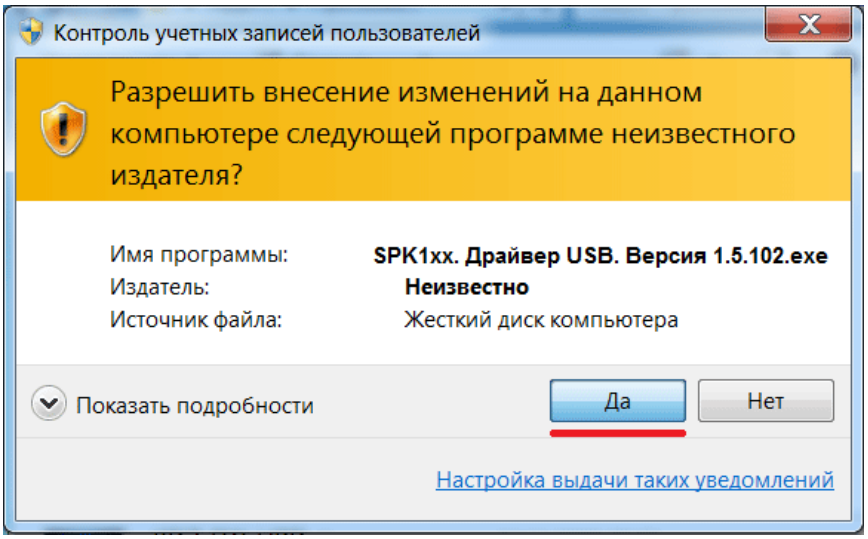


Рис. 5.15. Информационное сообщение при установке драйвера

Следует нажать **Да**.

Далее последовательно будут появляться диалоговые окна **Мастера установки драйверов СПК** и **Мастера установки драйверов устройств**, информационное сообщение **брандмауэра** Windows (если он включен) и диалоговые окна завершения установки:

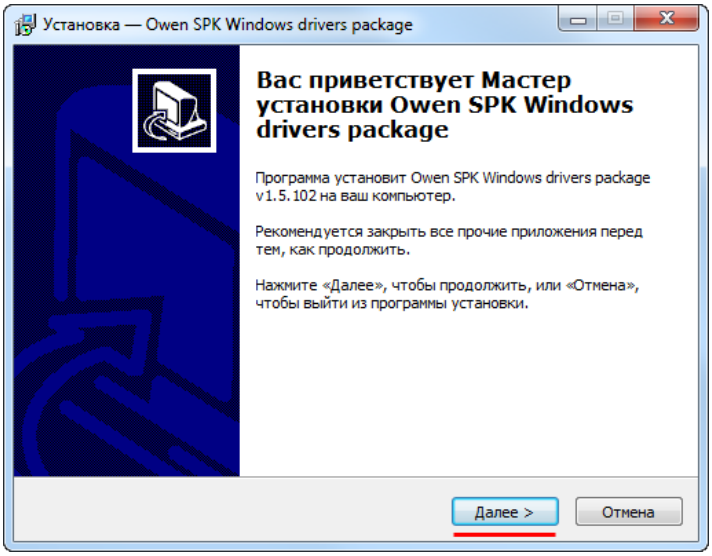


Рис. 5.16. Диалоговое окно **Мастера установки драйверов СПК**

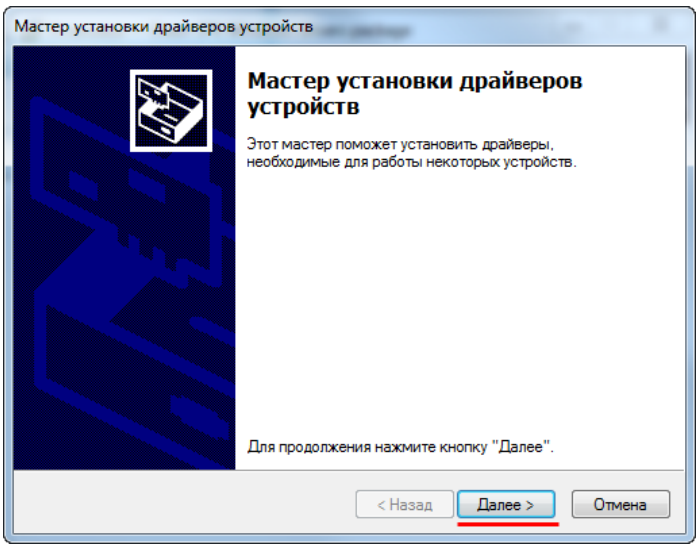


Рис. 5.17. Диалоговое окно **Мастера установки драйверов устройств**

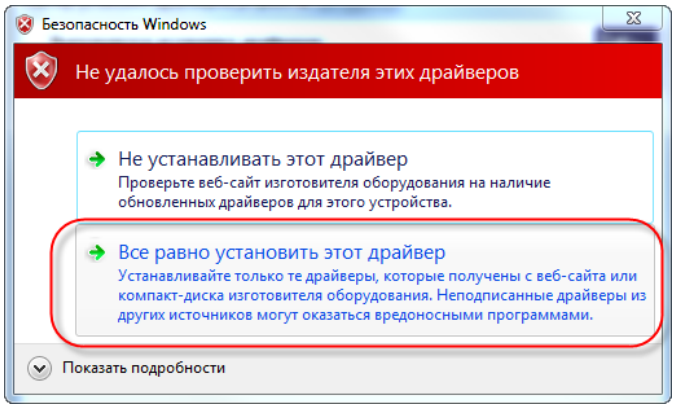


Рис. 5.18. Информационное сообщение брандмауэра Windows

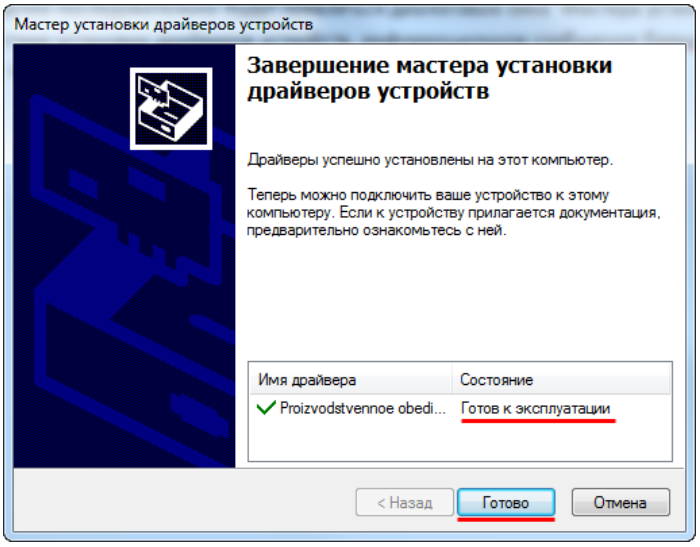


Рис. 5.19. Диалоговое окно завершения установки драйверов устройств

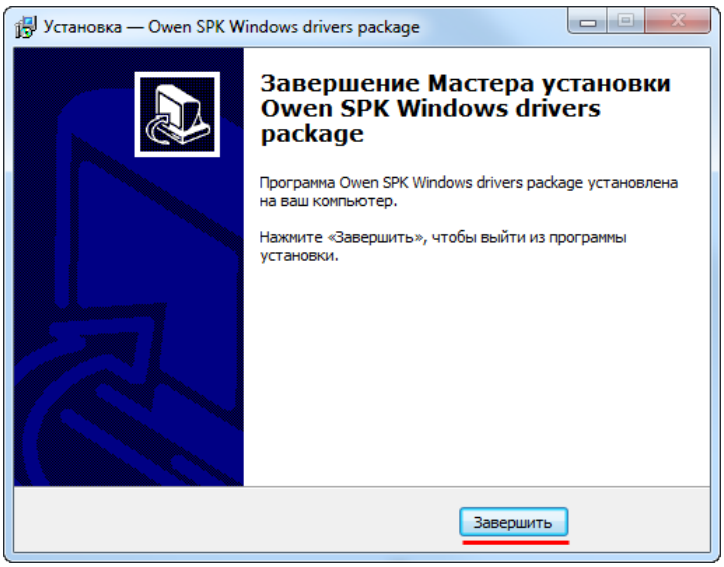


Рис. 5.20. Диалоговое окно завершения установки драйверов СПК

Теперь подсоединим конец В кабеля к USB В порту **включенного** контроллера, а конец А - к USB порту компьютера. ***В этот момент*** в Диспетчере устройств (**Пуск – Панель управления – Диспетчер устройств**) появится новый сетевой адаптер – **Owen SPK**.

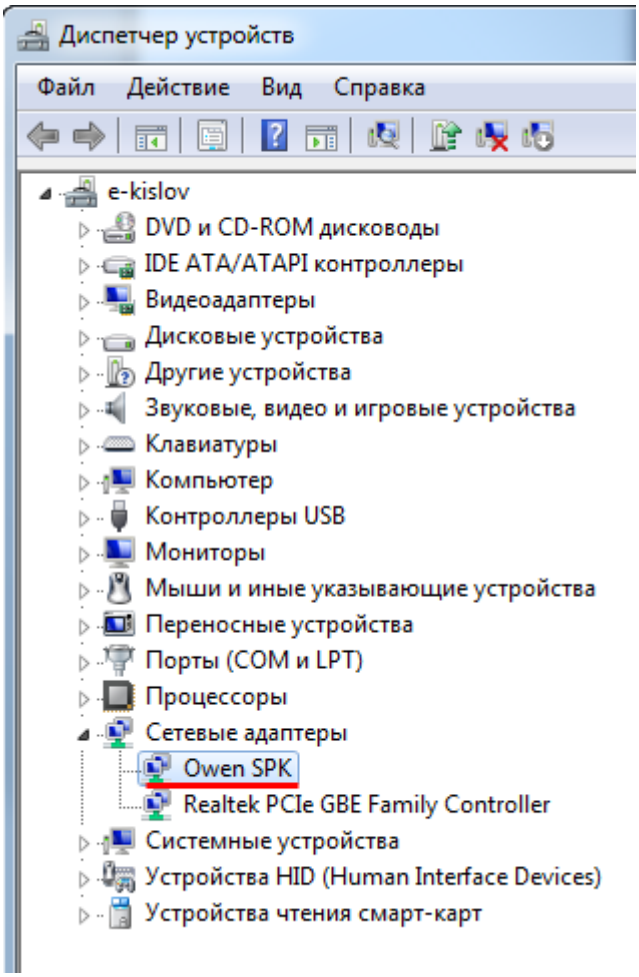


Рис. 5.21. Окно **Диспетчера устройств** после подключения СПК1xx

В меню Изменение параметров адаптера (**Пуск — Панель управления — Центр управления сетями и общим доступом – Изменение параметров адаптера**) появится новый виртуальный сетевой адаптер:

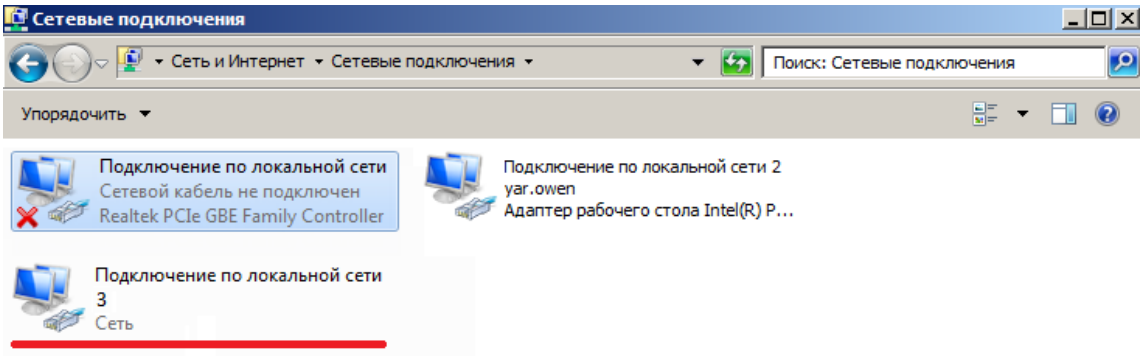


Рис. 5.22. Виртуальный сетевой адаптер для СПК 1xx

Настроим сетевой адаптер следующим образом (по аналогии с рис. 5.11):

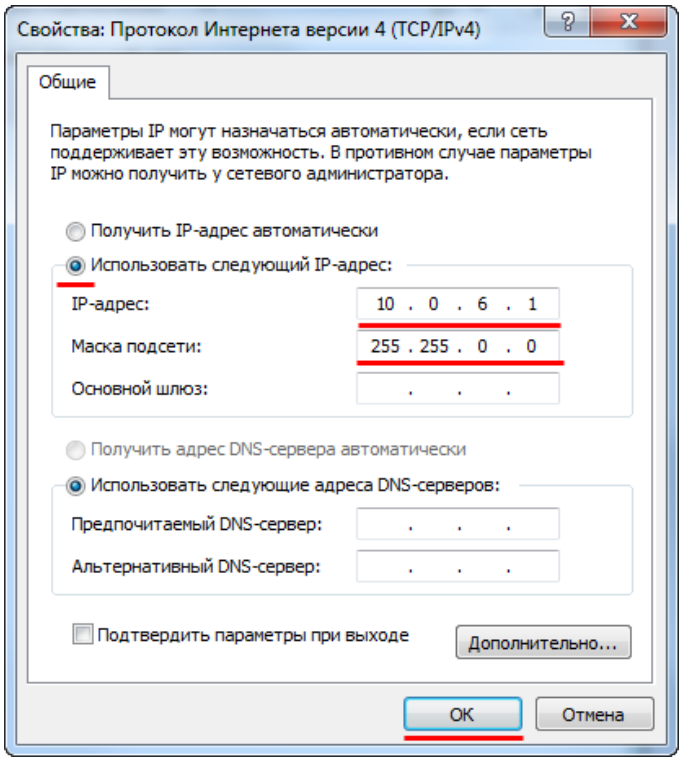


Рис. 5.23. Настройки виртуального сетевого адаптера для СПК 1xx

Зададим компьютеру IP-адрес **10.0.6.1** и маску **255.255.0.0** (***обратите внимание***: IP-адрес **совпадает** с IP-адресом **шлюза** контроллера, маска подсети **совпадает** с маской контроллера). Поля остальных настроек оставим пустыми. Нажмем **ОК**. Адаптеру потребуется несколько секунд, чтобы применить новые настройки.

Теперь мы можем проверить наличие связи между компьютером и контроллером. Откроем **командную строку** (**Пуск** — **Все программы** — **Стандартные** — **Командная строка**) и введем команду **ping 10.0.6.10 -t**. При наличии связи мы увидим следующий ответ:

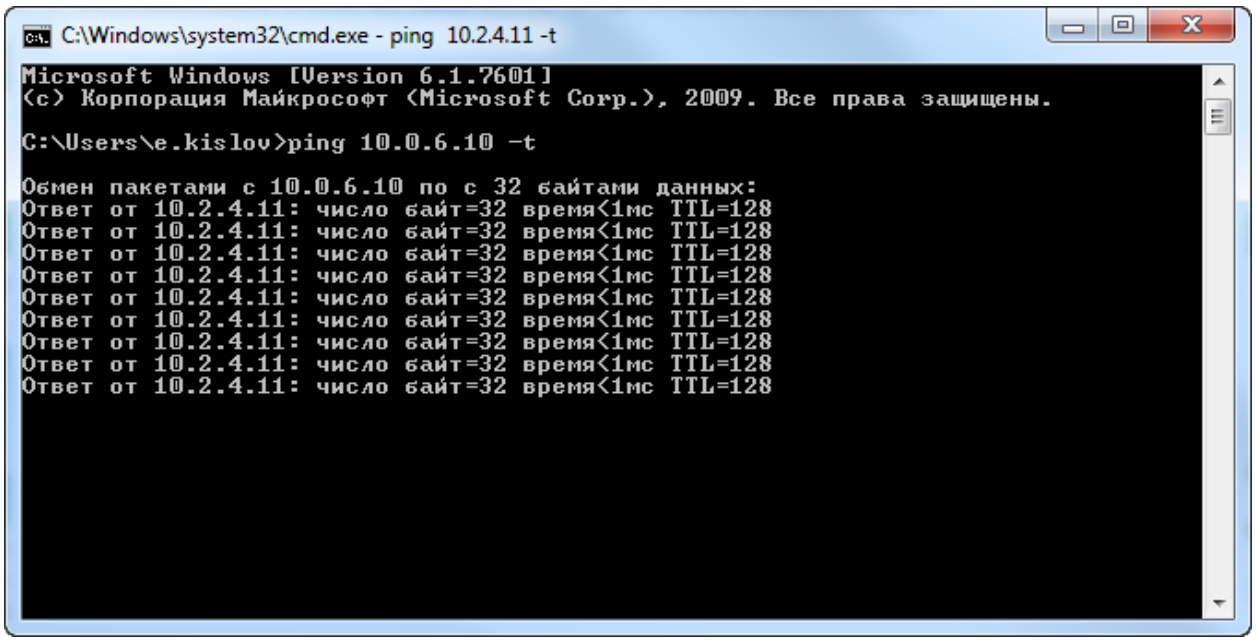


Рис. 5.24. Результат команды ping

Необходимо обратить внимание, что при перезагрузке контроллера или отключении питания необходимо обязательно ***переподключить*** кабель USB, т.е. отключить кабель от контроллера, дождаться его полной загрузки (которая занимает около 30 секунд), после этого подключить кабель к контроллеру.

5.3. Особенности настройки связи между СПК105 и ПК

Связь между контроллером СПК105 и пользовательским ПК осуществляется по USB с использованием кабеля типа **A – A**, входящего в комплект поставки:

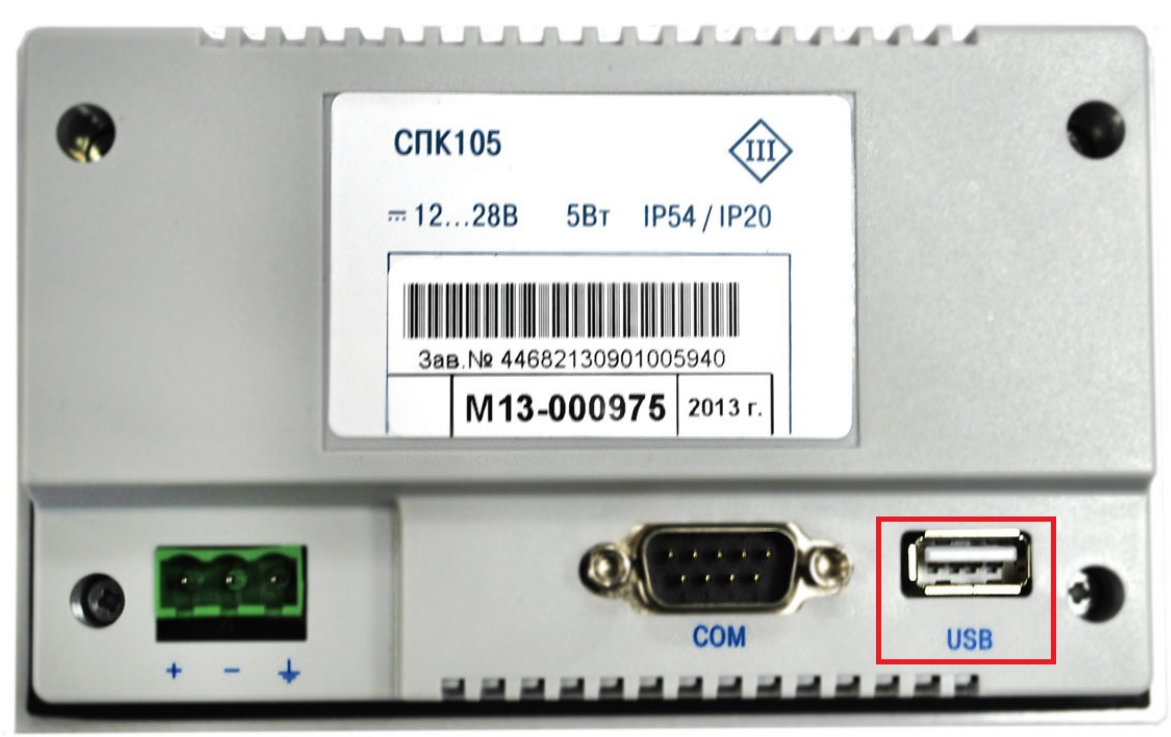


Рис. 5.25. Внешний вид задней панели СПК105

Рис. 5.26. Внешний вид кабеля USB A – A

Так как порт USB у СПК105 используется как для подключения контроллера к компьютеру, так и для подключения к контроллеру флэш-накопителя, перед выполнением первого или второго действия необходимо выбрать **режим работы** USB порта. Это нужно сделать в сервисном меню контроллера. Для того, чтобы открыть его, необходимо после включения питания ***до начала загрузки проекта*** контроллера коснуться экрана три раза.

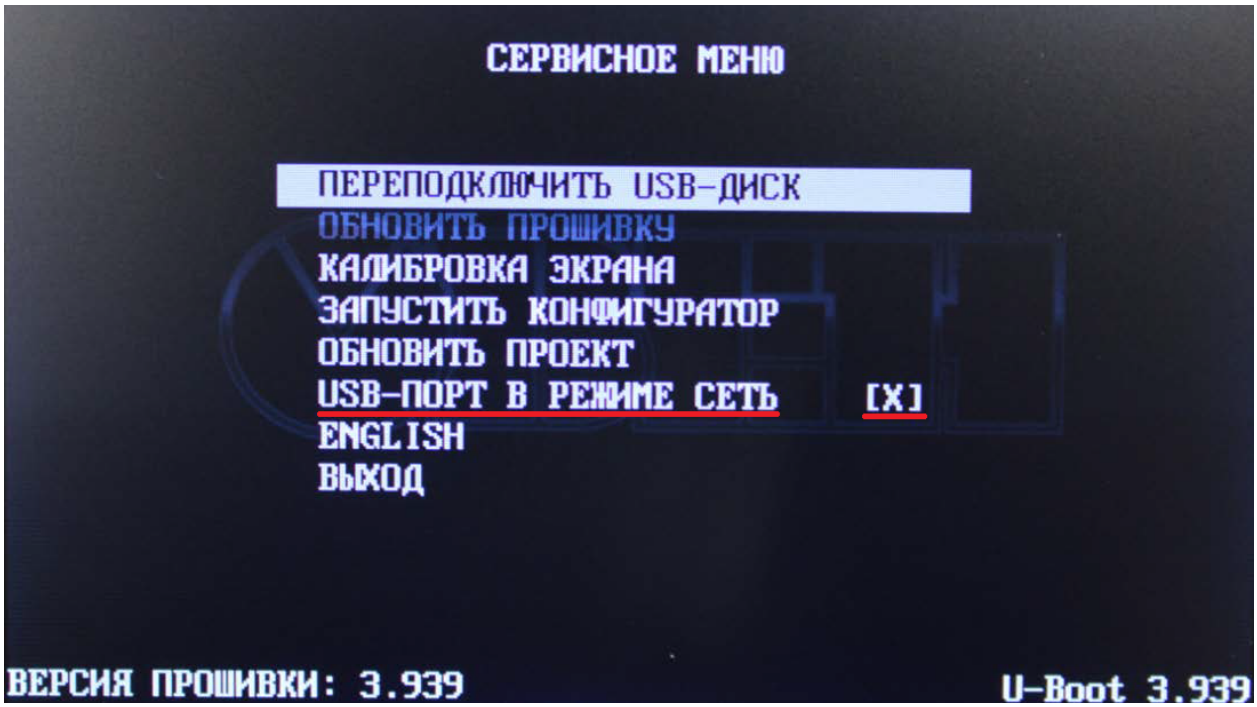


Рис. 5.27. Сервисное меню СПК105

В остальном процесс подключения СПК105 к компьютеру ***совершенно аналогичен*** процессу подключения СПК107/110 (см. п. 5.2).

Необходимо обратить внимание, что при перезагрузке контроллера или отключении питания необходимо обязательно ***переподключить*** кабель USB, т.е. отключить кабель от контроллера, дождаться его полной загрузки (которая занимает около 30 секунд), после этого подключить кабель к контроллеру.

5.4. Настройка связи контроллера и ПК в среде CODESYS

После того, как произведена настройка сетевых параметров контроллера и компьютера, необходимо установить связь между ними в среде CODESYS.

На этом этапе важно помнить про необходимость соответствия компонента **Device** (который определяется соответствующим **target-файлом**; см. п. 1.5, п. 3) и модели контроллера. При необходимости изменить тип устройства можно, выбрав на **Панели устройств** компонент **Device** и, нажав на него **ПКМ**, открыть окно **Обновить устройство**:

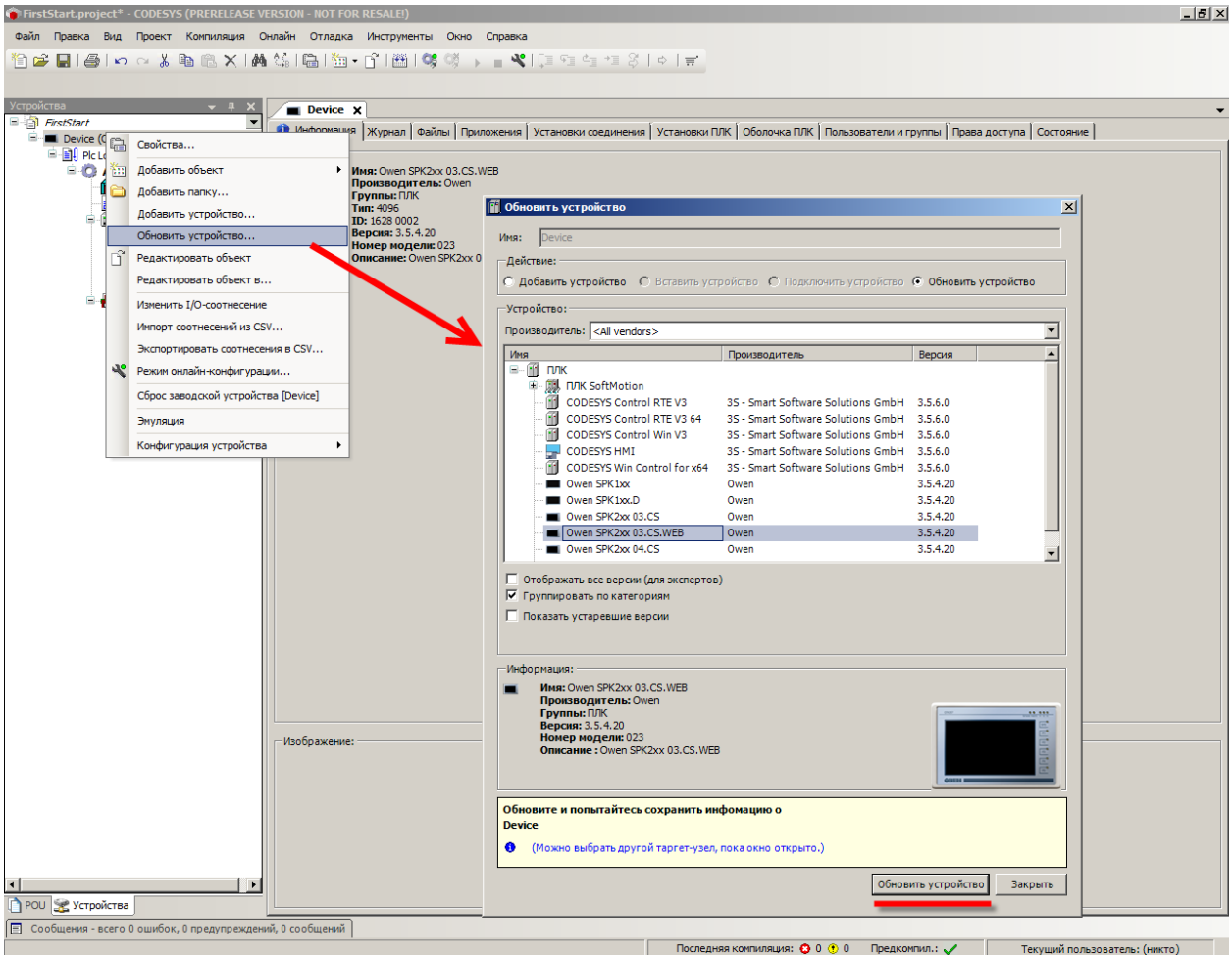


Рис. 5.28. Окно **Обновить устройство**

Теперь необходимо выбрать устройство, соответствующее имеющейся модели контроллера. **Название модели** указано на **задней крышке СПК** и в **конфигураторе (рис. 5.5., 5.6.)**. Следует помнить про ***необходимость соответствия*** версии прошивки контроллера, среды программирования **CODESYS** и **target-файла (см. п. 1.5)**. В нашем примере мы используем **СПК207.03.CS.WEB** и поэтому выбираем из списка соответствующее устройство.

После выбора устройства следует нажать кнопку **Обновить устройство** и закрыть окно. На **Панели устройств** у компонента **Device** отобразится название выбранного **устройства.**

Теперь необходимо произвести настройку **gateway** (**шлюза**).

Двойным нажатием **ЛКМ** по компоненту **Device** (или одиночным нажатием по вкладке вверху рабочей области) перейдем к его настройкам. Откроем вкладку **Установки соединения** и нажав на кнопку **gateway** выберем пункт **Add new gateway**:

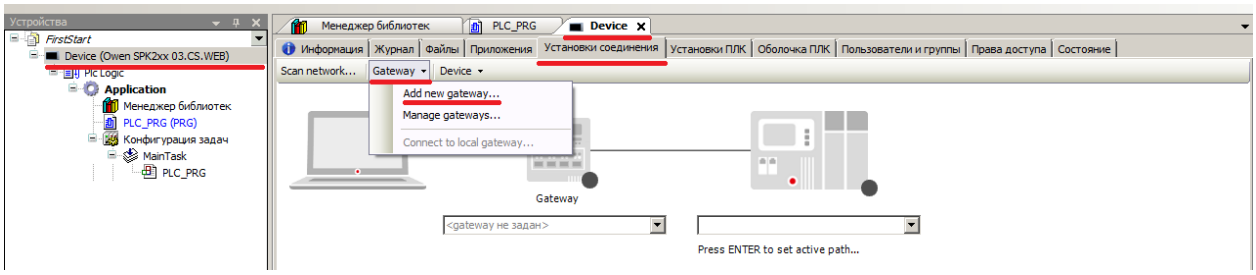


Рис. 5.29. Создание нового gateway (шлюза)

Настройки оставим по умолчанию (имя – **Gateway-1**, IP-адрес – **localhost**). Закроем окно настройки шлюза и нажмем кнопку **Scan network** . В появившемся списке выберем наш контроллер и установим связь, нажав **ОК**.

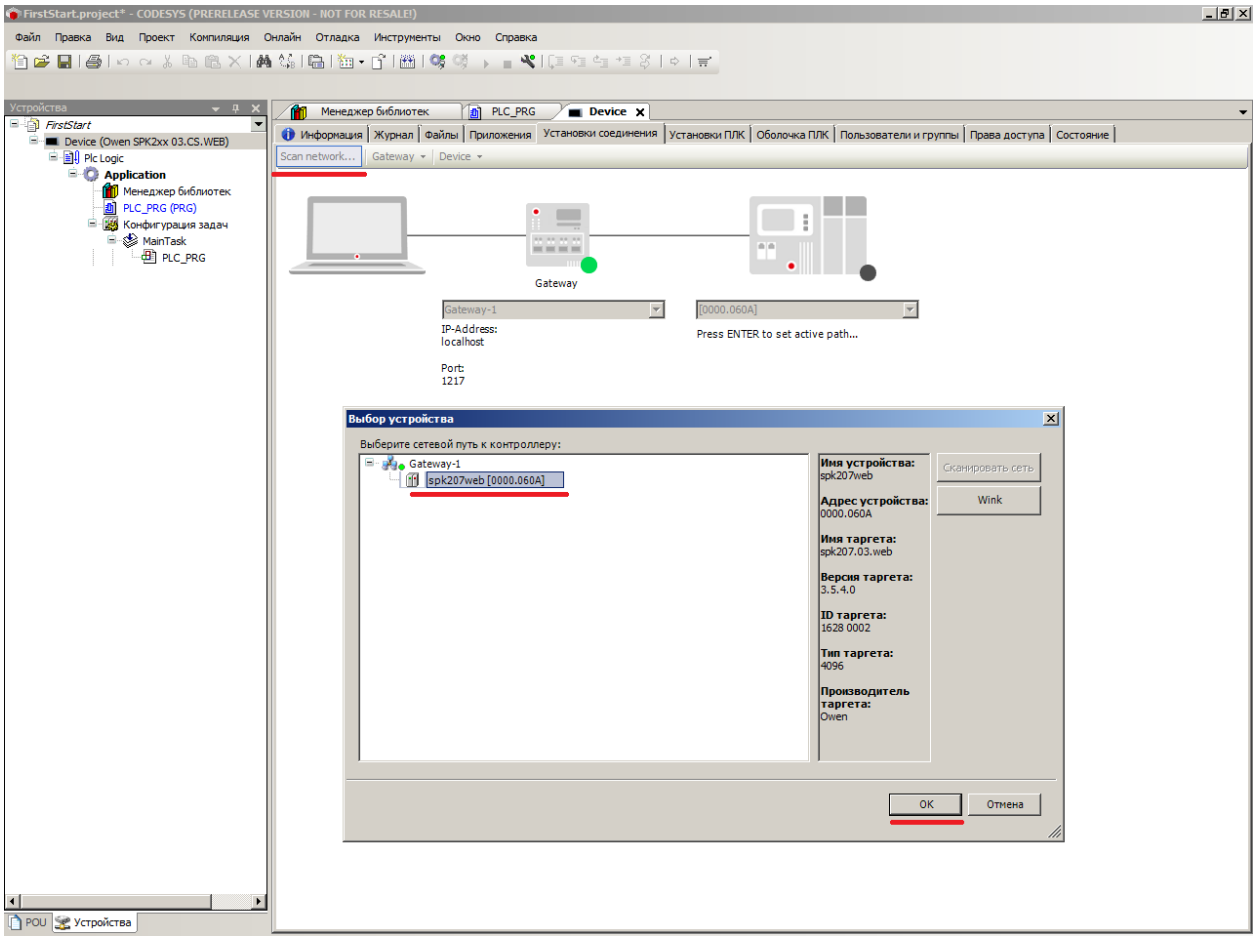


Рис. 5.30. Окно сканирования сети

В случае успешной установки связи индикаторы шлюза и контроллера загорятся зеленым:

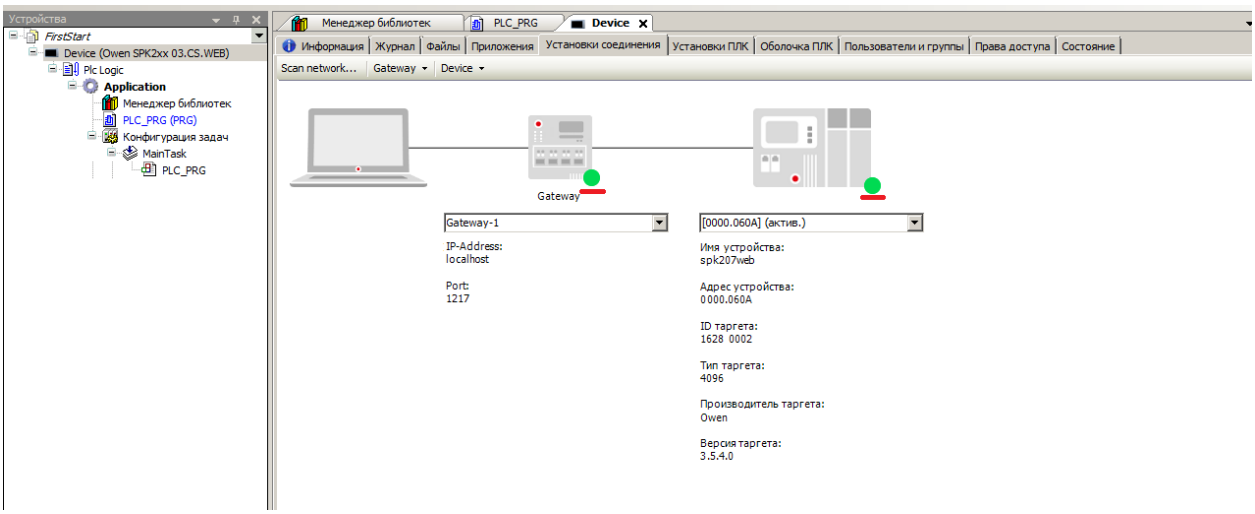


Рис. 5.31. Результат успешной установки связи

6. Загрузка и запуск «пустого» проекта

Итак, после выполнения п. 3 и п. 5 мы имеем пустой проект и установленную связь между компьютером и контроллером. Но загрузка пустого проекта не вызовет в контроллере видимых изменений. Поэтому для наглядности добавим в проект компонент **Визуализация** с названием по умолчанию:

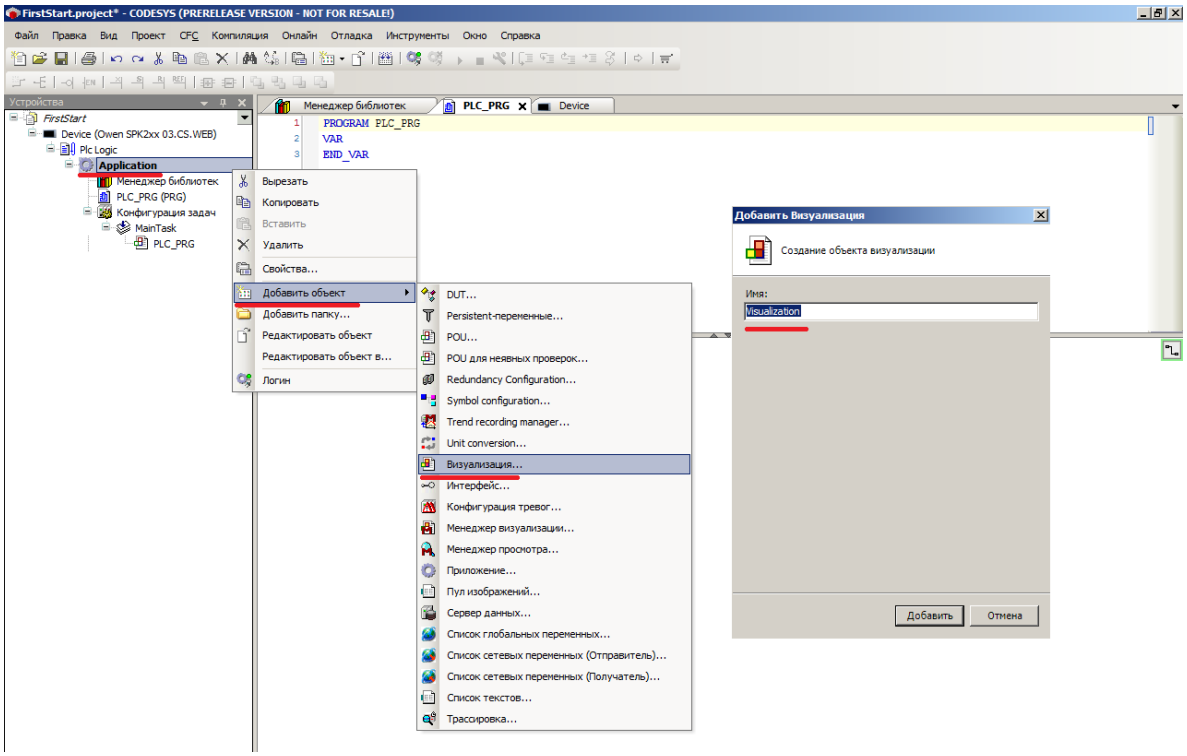


Рис. 6.1. Добавление компонента **Визуализация**

В результате **Панель устройств** примет следующий вид:

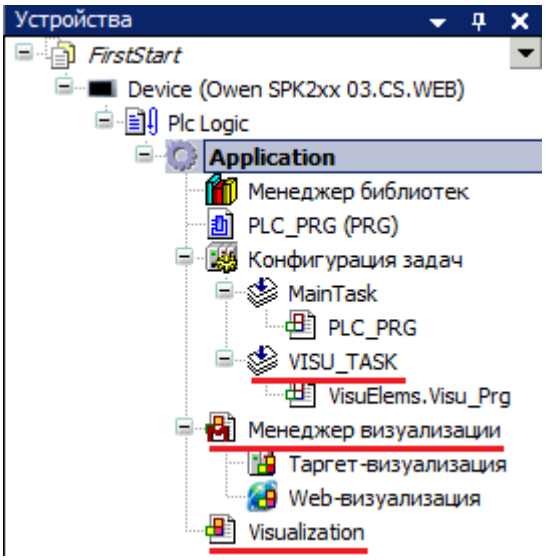


Рис. 6.2. Внешний вид **Панели устройств** после добавление компонента **Визуализация**

Теперь наш проект имеет пустой экран визуализации. Как можно заметить, вместе с компонентом **Визуализация** добавился компонент **Менеджер визуализации** и новая **задача** с названием **VISU_TASK**. Подробнее все компоненты **Панели устройств** будут рассмотрены в п. 7.

Для загрузки проекта в **оперативную память** контроллера необходимо в меню **Онлайн** нажать кнопку **Логин** (она также продублирована на **Панели инструментов**):

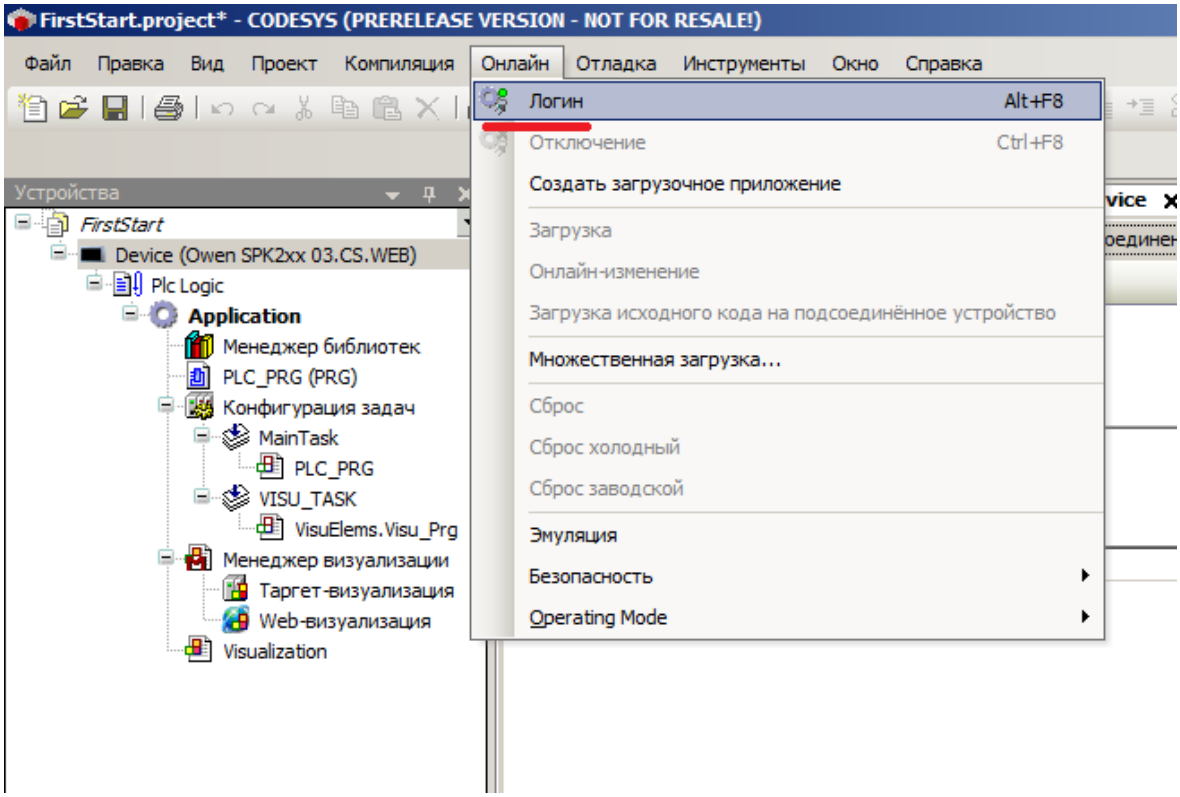


Рис. 6.3. Кнопка **Логин**

Поскольку в контроллер уже загружен демонстрационный проект, то появится следующее диалоговое окно:

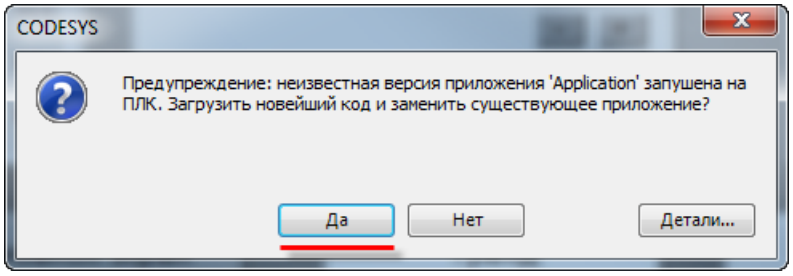


Рис. 6.4. Диалоговое окно загрузки проекта

Нажимаем **Да**. ***Нужно понимать***, что демонстрационный проект при этом **будет удален.**

Теперь проект загружен в **оперативную память** контроллера, информация из которой стирается при отключении питания. Для того, чтобы проект оставался в памяти контроллера после перезагрузки, необходимо произвести его загрузку во **flash-память**. Flash-память имеет

ограничение на число циклов перезаписи, поэтому ***рекомендуется*** на этапе разработки и тестирования проекта загружать его именно в оперативную память. ***При необходимости*** загрузки проекта во **flash-память** следует воспользоваться командой **Создать загрузочное приложение** из меню **Онлайн**.

После загрузки проекта в оперативную память, **Панель устройств** будет выглядеть следующим образом:

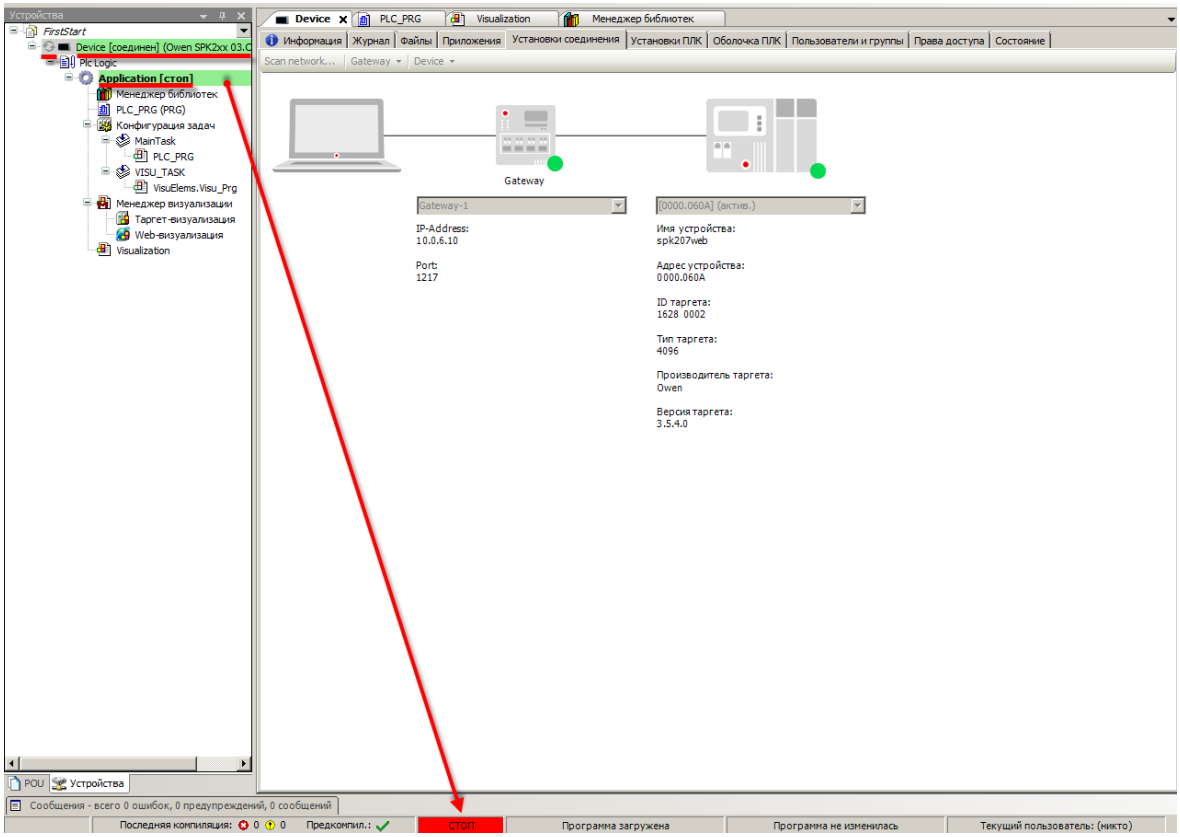


Рис. 6.6. Панель устройств загруженного проекта

По умолчанию приложение загруженного проекта ***не запущено*** (***остановлено***)**.** Информация об этом продублирована в **Строке состояния**. Компоненты **Device** и **Application** подсвечены зеленым, что характеризует установку связи и наличие пользовательского приложения в памяти контроллера.

Для запуска проекта следует нажать кнопку **Старт**, расположенную в меню **Отладка** (она также продублирована на **Панели инструментов**):

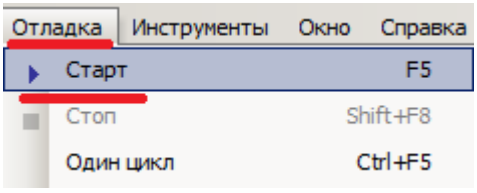


Рис. 6.7. Кнопка запуска проекта

После этого изображение на дисплее контроллера сменится на пустой белый экран, который мы добавили в проект в начале пункта, а **Панель устройств** и **Панель сообщений компиляции** будут выглядеть следующим образом:

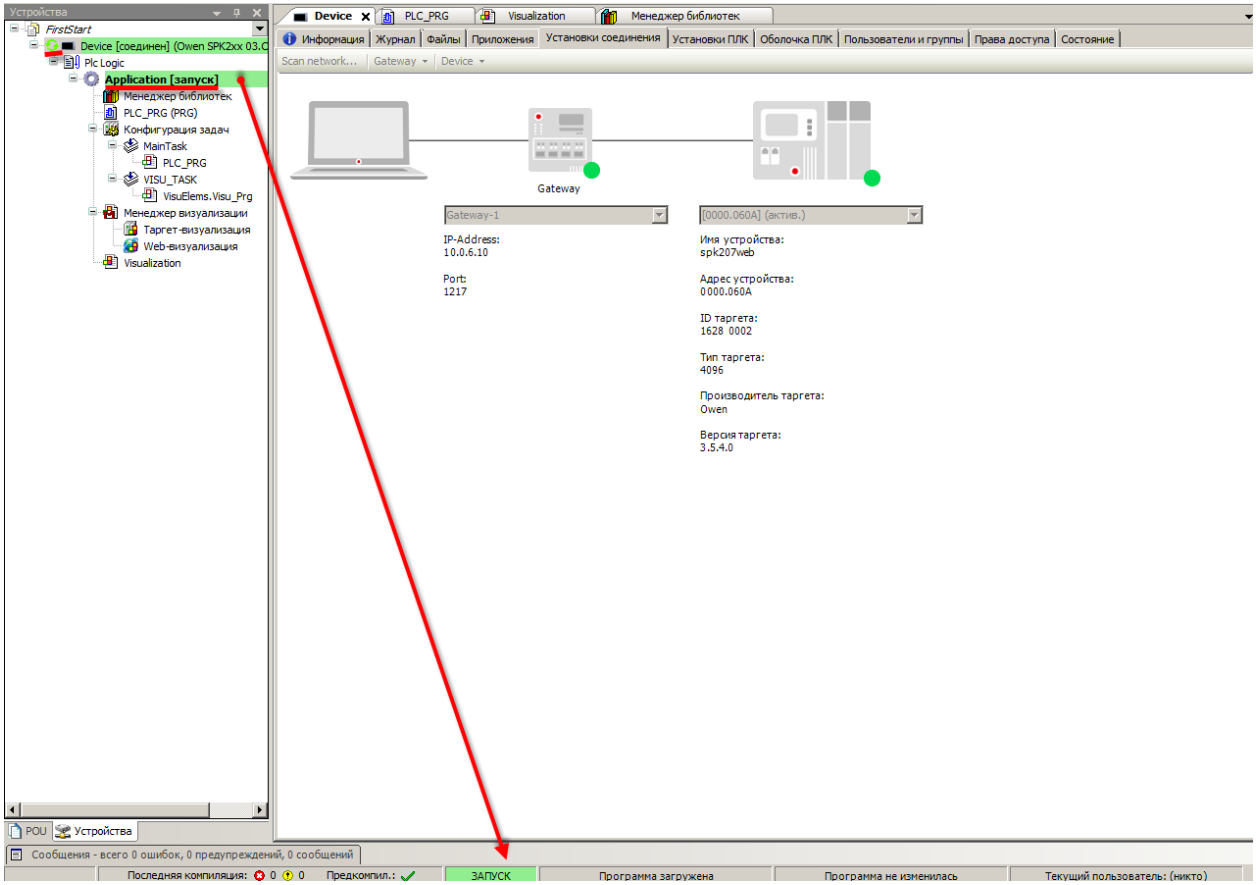


Рис. 6.8. Панель устройств запущенного проекта

У пользователя есть возможность загружать проект в контроллер с подключенного к нему **USB-накопителя**. Подробно этот процесс рассмотрен в Руководстве по эксплуатации.

7. Создание пользовательского проекта

7.1. Постановка задачи

Рассмотрим процесс разработки пользовательского проекта в среде CODESYS на примере управления значением **температуры** с помощью **двухпозиционного регулятора**.

Двухпозиционный регулятор (компаратор) сравнивает значение **измеренной величины** с **эталонным (уставкой**). Состояние выходного дискретного сигнала изменяется на противоположное, если входной сигнал (измеренная величина) пересекает пороговый уровень (уставку). Обычно двухпозиционный регулятор имеет задаваемую **зону гистерезиса** (Δ), которая используется для предотвращения «дребезга» (постоянного включения/выключения) управляющего выходного устройства (например, реле) вблизи значения уставки.

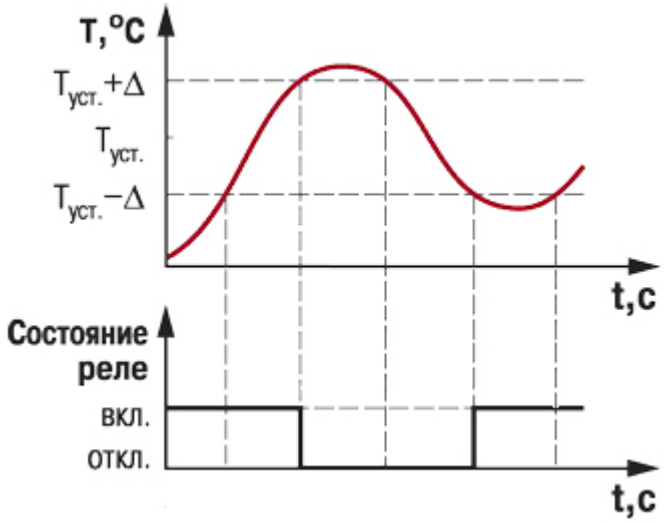


Рис. 7.1. Принцип действия двухпозиционного регулятора температуры

***Сформулируем следующую задачу*:**

Температура в помещении измеряется **термодатчиком** и контролируется **кондиционером**. Необходимо создать **автоматическую систему управления температурой с двухпозиционным регулятором**. Система должна обладать **графическим интерфейсом** со следующим функционалом:

- отображение текущего значения температуры;

- **ручной ввод** текущего значения температуры (используется в **режиме эмуляции**);

- настройка параметров двухпозиционного регулятора (ввод значения уставки по температуре и значения гистерезиса);

- возможность включения/выключения кондиционера;

- индикация режима работы кондиционера;

- ввод верхней и нижней **аварийной уставки** тревог по температуре;

- индикация выхода значения температуры за одну из аварийных уставок;

- построение **тренда** по температуре, уставкам гистерезиса, уставкам тревог;

- ведение **Журнала событий** с сигналами о выходе температуры за значения гистерезиса и аварийных уставок.

Система регулирования, построенная на базе **контроллера СПК2xx**, будет иметь **один входной аналоговый сигнал** (температура) и **два выходных дискретных** - состояние кондиционера (включен/выключен) и режим его работы (обогрев помещения/охлаждение помещения). Для ввода-вывода сигналов будут использованы **модули ОВЕН серии Мх**: МВ110-224.2А и МУ110-224.8Р. Связь между контроллером и модулями осуществляется по протоколу **Modbus RTU** (см. п. 7.8). Для наглядности мы также создадим в контроллере модель кондиционера для **эмуляции процесса регулирования** в случае отсутствия реальных сигналов.

Нужно отметить, что целью главы является не демонстрация эффективного решения конкретной задачи, а описание общих принципов создания проекта в CODESYS с попыткой охватить как можно больший функционал среды программирования, что отражается на искусственности и нецелесообразности отдельных моментов.

7.2. Структура проекта. Общие аспекты создания проекта

Проект в CODESYS имеет иерархическую модульную структуру и состоит из отдельных узлов и их компонентов, расположенных на **Панели Устройств**. В этой структуре можно выделить несколько уровней:

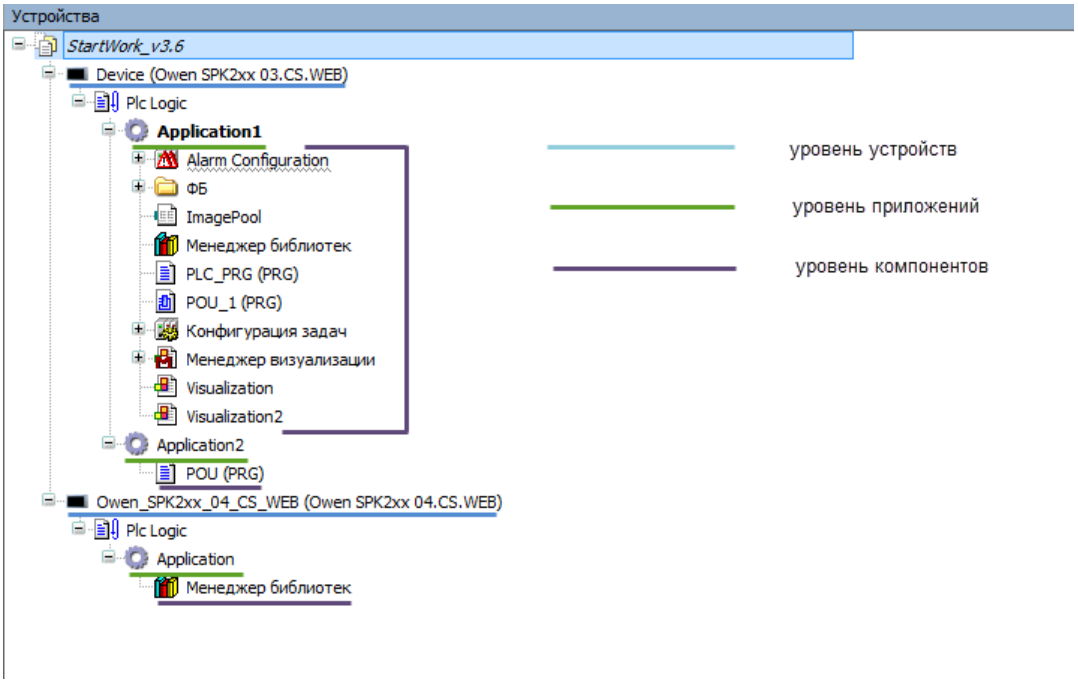


Рис. 7.2. Структура **Панели устройств**

1. **Device** - пользовательское устройство (например, контроллер). По умолчанию название устройства выглядит как **Device (Модель устройства),** где Device - имя устройства, которое может быть изменено пользователем; название модели устройства, приведенное в скобках, определяется **target-файлом** и не может быть изменено. В проекте может быть несколько устройств. Каждое из устройств является либо **программируемым**, либо **параметрируемым (**на данный момент компании ОВЕН ***не производит*** параметрируемые устройства, поддерживающие CODESYS V3.x**)**. Тип устройства также определяется target-файлом. Программируемые устройства содержат дополнительный узел **Plc_logic**;

2. **Application** - приложение, которое будет запускаться на устройстве. Имя приложения может быть изменено пользователем. Для каждого устройства может быть создано **несколько** приложений, но **активным** из них в каждый момент времени является только одно. Соответственно, невозможно запустить несколько приложений **одновременно** и загрузить в контроллер более одного приложения;

3. **Компоненты** - отдельные модули, из которых состоит приложение. Часть из них автоматически создается одновременно с проектом, остальные по необходимости добавляются пользователем. Компоненты могут содержать **дочерние компоненты**. Имена некоторых (но не всех) компонентов могут быть изменены пользователем.

По умолчанию созданный проект содержит одно приложение, которое включает три компонента:

1. Пустую программу с название **PLC_PRG** (**язык** программы определяется на этапе создания проекта, см. п.3);

2. Компонент **Конфигурация задач**, содержащий дочерний **компонент** задачу с названием **MainTask**, к которой привязана программа **PLC_PRG**. Задача определяет частоту и правила вызова привязанного к ней **POU**;

3. Компонент **Менеджер библиотек**, отвечающий за подключение к проекту внешних библиотек, которые добавляют дополнительный функционал среды программирования.

В целом, обычно процесс создания проекта состоит из следующих этапов:

1. Создание экранов визуализации;

2. Разработка пользовательских программ и алгоритмов;

3. Настройка требуемых компонентов (например, Конфигурации задач);

4. Настройка обмена данными между контроллером и другими устройствами (например, модулями ввода-вывода или датчиками), связь переменных из пользовательских программ с соответствующими реальными сигналами.

Последовательность этапов разработки определяется пользователем; в данном руководстве мы будем придерживаться приведенной выше схемы.

7.3. Создание экранов визуализации

7.3.1. Предварительная настройка

Для решения сформулированной в п. 7.1 задачи нам потребуются три экрана визуализации:

1. Основной экран (отображение информации и управление);
2. Экран тренда;
3. Экран Журнала тревог.

В данный момент в нашем проекте имеется лишь один экран визуализации с названием **Visualization**, созданный в п. 5. Переименуем его в **MainScreen** (использование **кириллицы** в названиях компонентов не поддерживается) с помощью двойного нажатия **ЛКМ** на название экрана; после этого создадим еще два экрана с названиями **Trend** и **Alarms_log**:

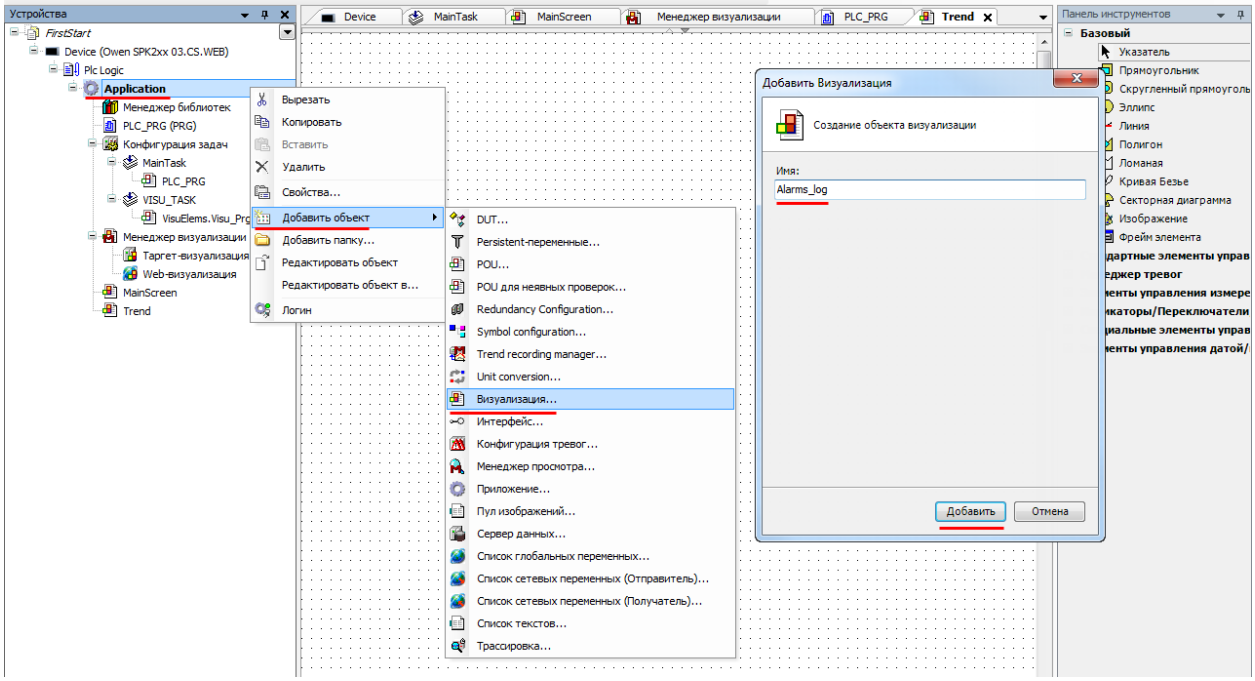


Рис. 7.3. Добавление экрана визуализации

Перед тем, как начать заниматься наполнением экранов графическими примитивами, следует настроить компонент **Менеджер визуализации** и его дочерние компоненты **Target-визуализация** и **Web-визуализация**.

Рассмотрим настройки вкладки **Установки** Менеджера визуализации:

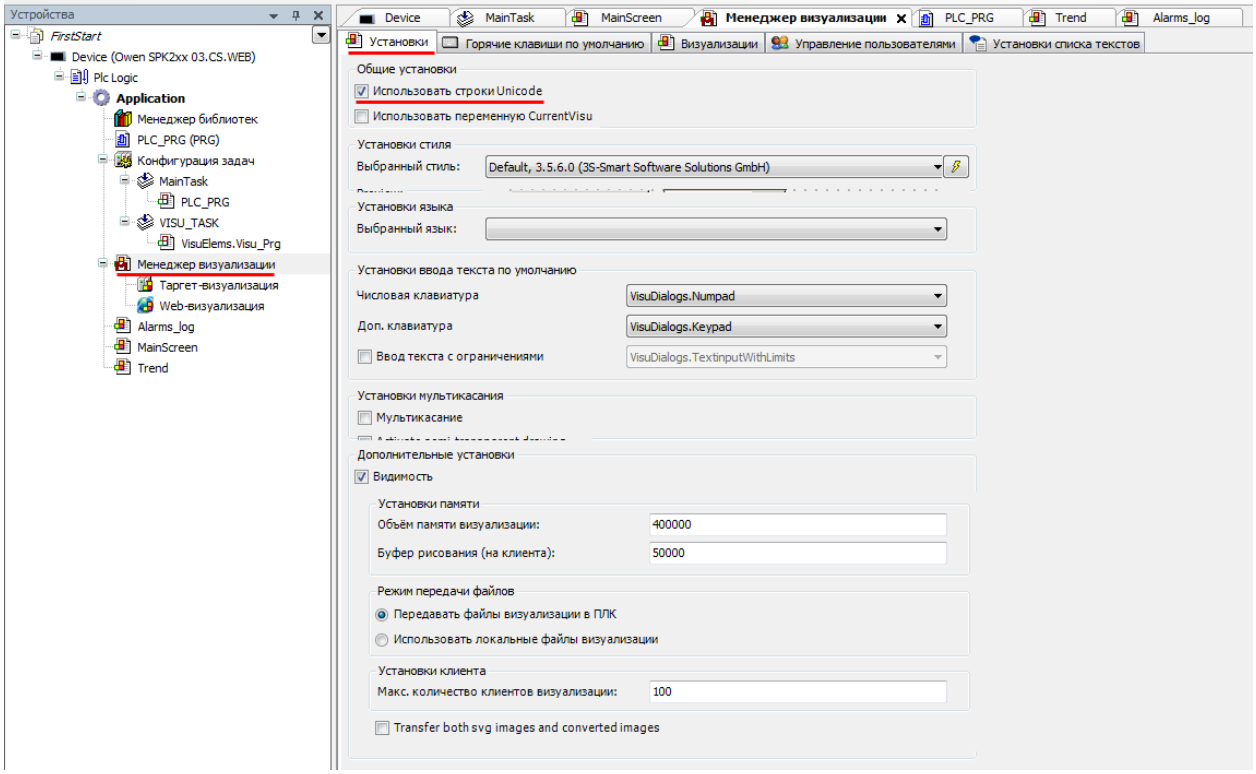


Рис. 7.4. Установки **Менеджера визуализации**

Для отображения в текстах визуализации **кириллицы** поставим галочку **Использовать строки Unicode**. Вторая опция – **Использовать переменную CurrentVisu** – добавляет в проект **одноименную строковую переменную**, которая определяет, какой из экранов отображается в данный момент. Соответственно, записывая в нее названия экранов визуализации, можно осуществлять переключения между ними. В нашем примере работа этой переменной ***рассмотрена не будет***.

Остальные настройки касаются стиля визуализации (который определяет внешний вид графических примитивов), выбора стартового языка визуализации (при реализации мультиязычной визуализации), используемых экранных клавиатур, объема памяти визуализации и т.д. Оставим их в значениях ***по умолчанию***.

Настроим компонент **Target-визуализация**, который является дочерним компонентом **Менеджера визуализации**:

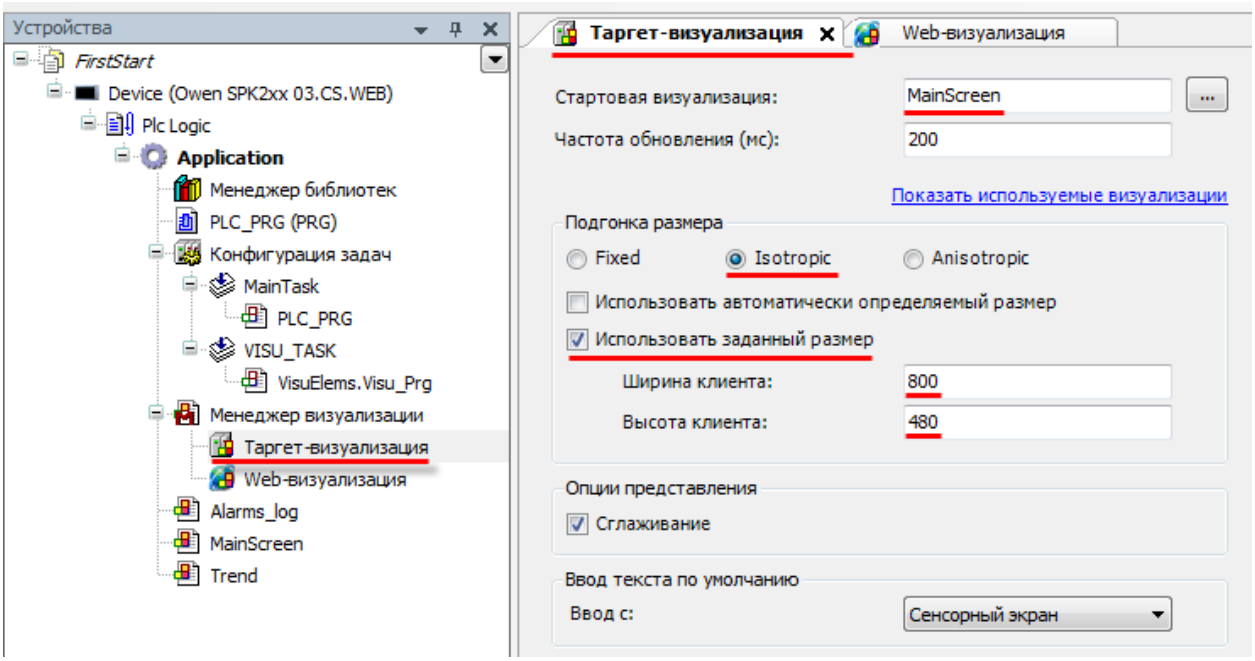


Рис. 7.5. Настройки target-визуализации

Имя стартовой визуализации должно соответствовать названию экрана, который будет отображаться при запуске проекта. В нашем примере это **MainScreen**.

Опция **Подгонка размера** определяет размер визуализации:

1. **Fixed** – отображается часть визуализации заданного размера;

2. **Isotropic** – позволяет задавать размер визуализации в пикселях, при этом при изменении размеров экрана, экран визуализации будет ***деформироваться с сохранением*** соотношения сторон;

3. **Anisotropic** – позволяет задавать размер визуализации в пикселях, при этом при изменении размеров экрана, экран визуализации будет ***деформироваться без сохранения*** соотношения сторон.

Рекомендуется использовать вариант **Isotropic** с указанием **разрешения экрана** СПК. В нашем примере используется **СПК207** с разрешением экрана **800x480**.

Опции **web-визуализации** практически идентичны target-визуализации. Поскольку web-визуализация запускается на внешних устройствах, ее разрешение должен соответствовать разрешению этих устройств. Например, если в качестве устройства просмотра web-визуализации используется **Full HD** монитор, то рекомендуется указывать разрешение **1920x1080**. Для того, чтобы указать разрешение, необходимо выбрать режим **Fixed**; ***после указания разрешения следует выбрать режим Isotropic.***

Имя **.htm-файла** определяет название web-страницы визуализации.

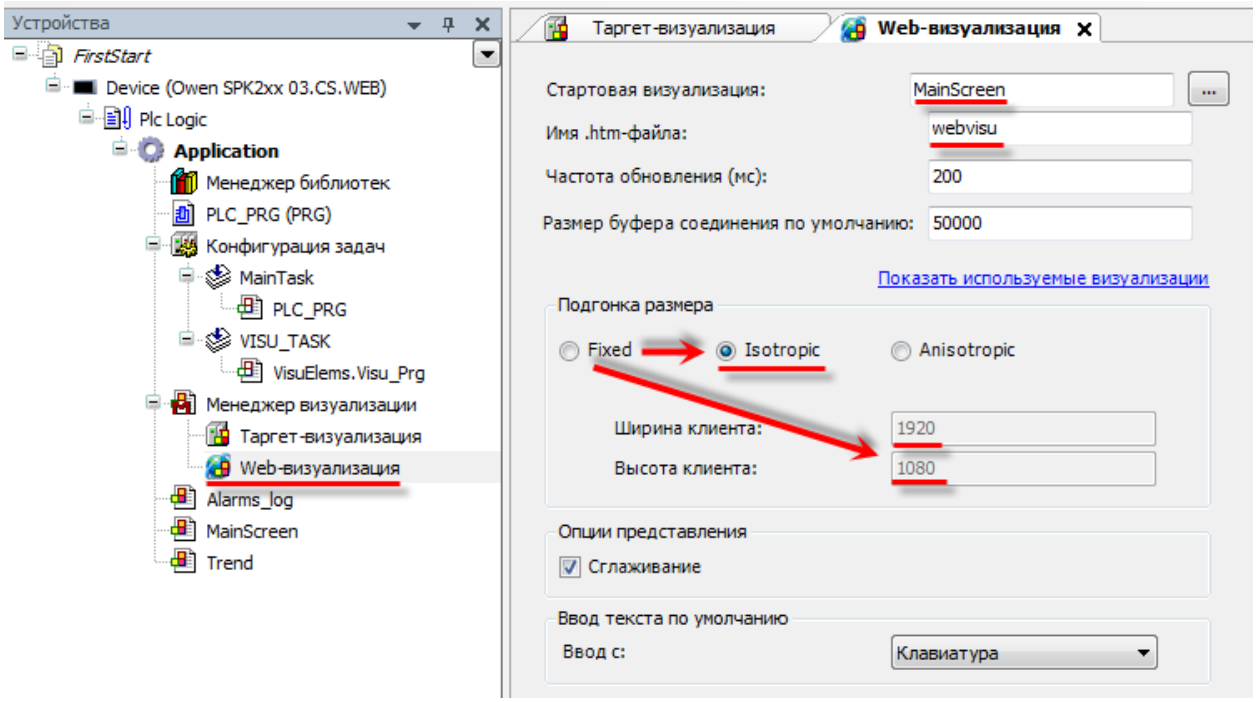


Рис. 7.6. Настройки web-визуализации

Помимо настройки разрешения target- и web-визуализаций необходимо установить **разрешение каждого экрана** визуализации. Для этого следует нажать **ПКМ** на название соответствующего экрана и выбрать пункт **Свойства**, вкладка **Визуализация**:

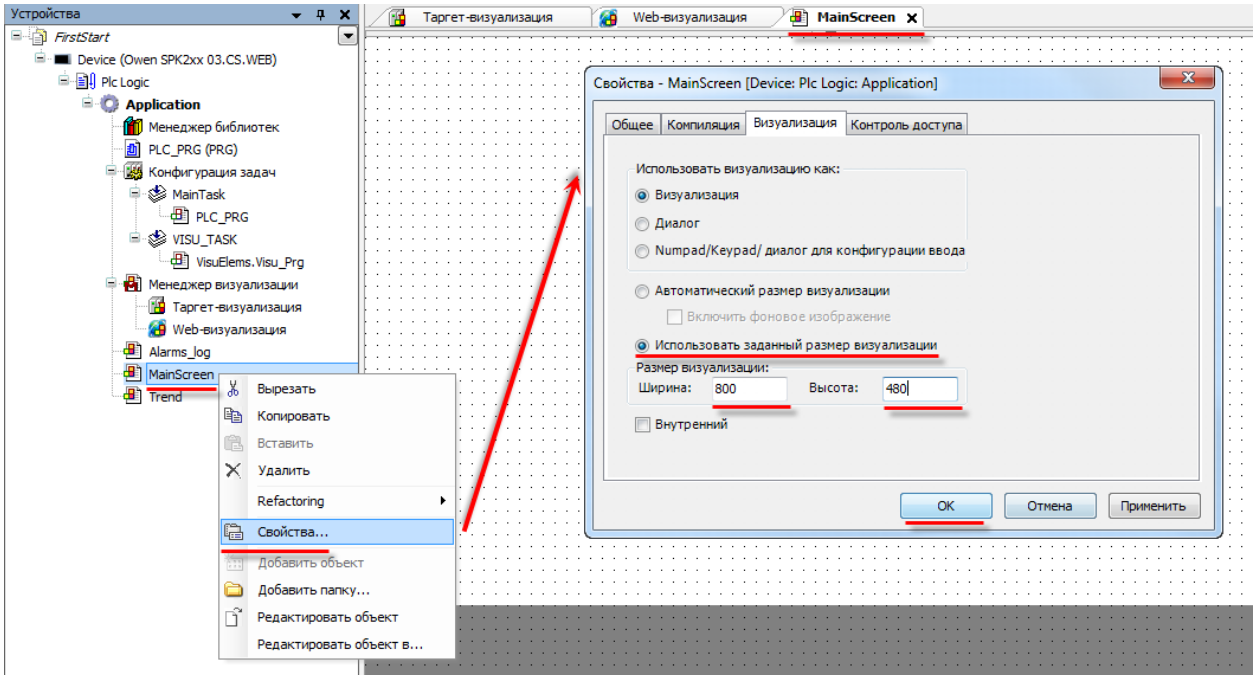


Рис. 7.7. Свойства экрана визуализации **MainScreen**

Также перед началом разработки экранов визуализации рекомендуется в **Опциях** (**Инструменты – Опции**) включить **сетку**:

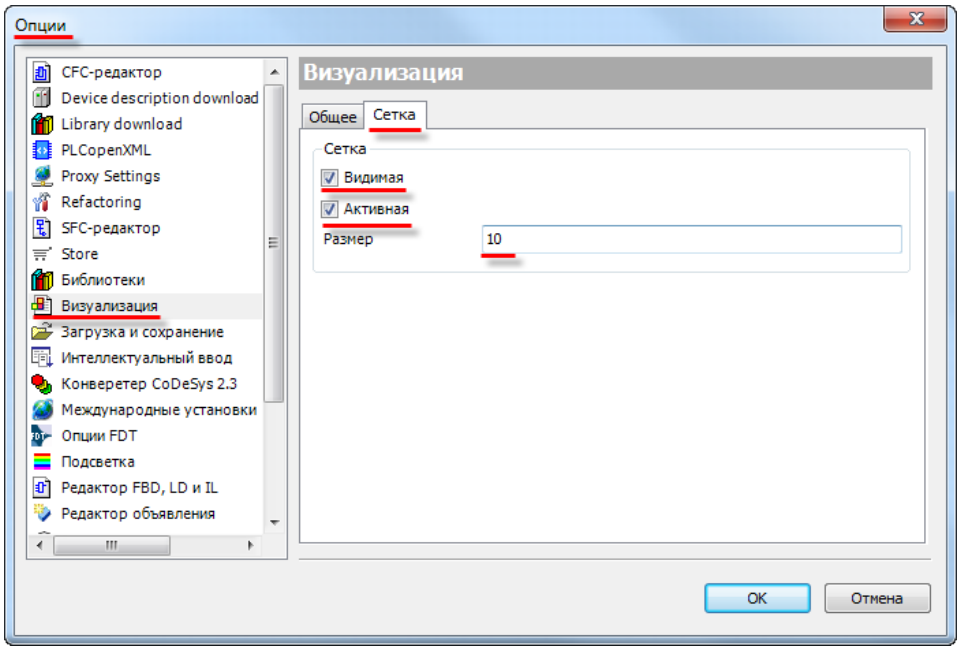


Рис. 7.8. Настройки сетки

7.3.2. Наполнение экрана MainScreen

Экран **MainScreen** будет использоваться для отображения текущего значения температуры, переключения режима измерения, установки значения температуры в режиме «Эмуляция (Ручной ввод)», задания уставки температуры и гистерезиса, управления кондиционером и индикации режима его работы.

Двойным нажатием **ЛКМ** на компонент **MainScreen** откроем **Редактор визуализации** соответствующего экрана:

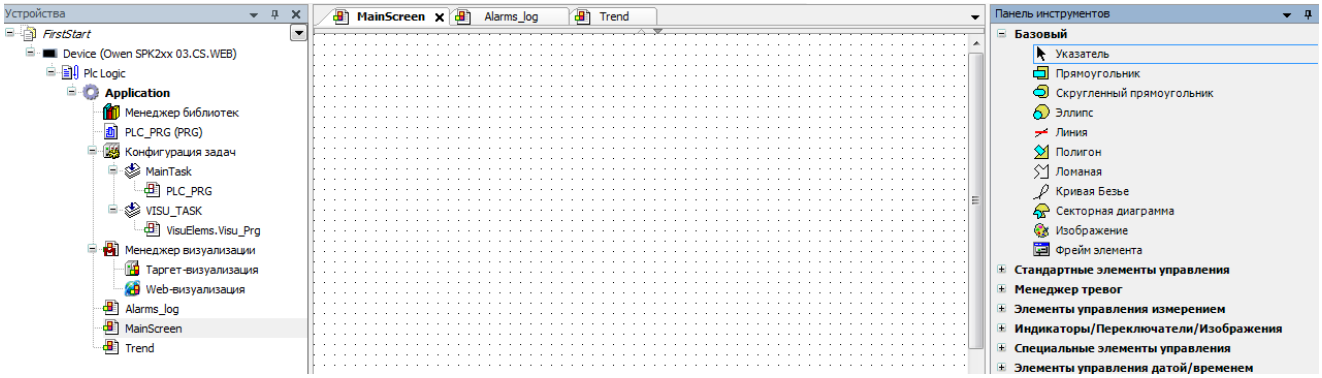


Рис. 7.9. Редактор визуализации

Справа от **рабочего поля** расположена **Панель инструментов редактора**, содержащая набор **графических примитивов**; выделив **элемент**, можно с помощью одиночного нажатия **ЛКМ** разместить его на рабочем поле.

Далее будет последовательно описан процесс наполнения экранов визуализации элементами и их графическая настройка. ***Функциональная настройка элементов и привязывание к ним переменных будут описаны в п. 7.5.***

I. Элемент Кнопка

Начнем наполнение экрана **MainScreen** с создания кнопок перехода на другие экраны. Выделим во вкладке **Стандартные элементы управления** элемент **Кнопка** и разместим его на рабочем поле:

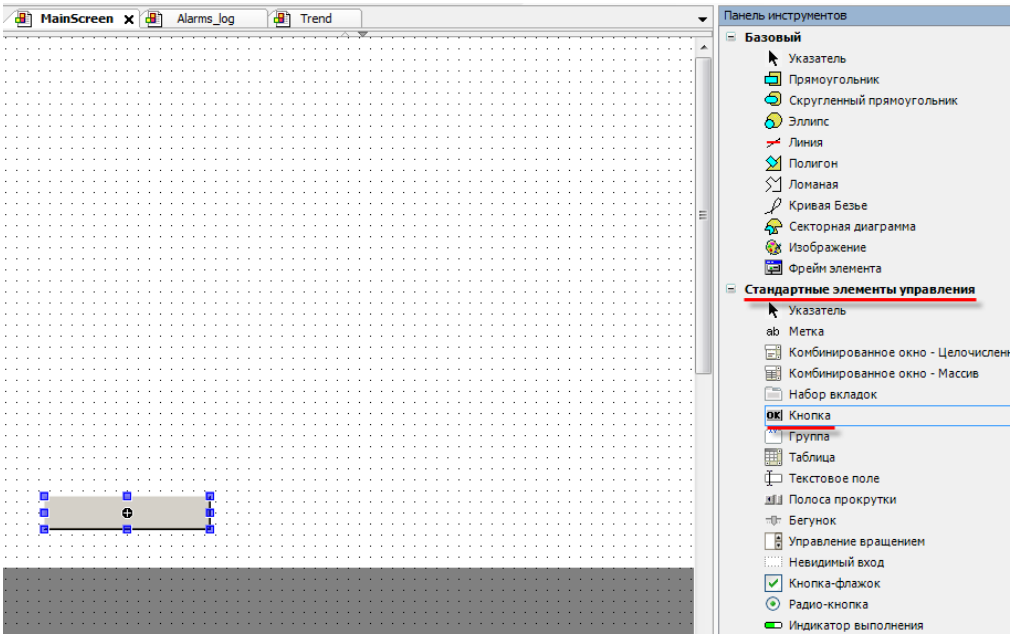


Рис. 7.10. Создание кнопки

Выделив созданную кнопку нажатием **ЛКМ** (для этого достаточно нажать **ЛКМ** на любую точку рабочего поля, а затем обратно на кнопку), мы откроем ее **Панель свойств**:

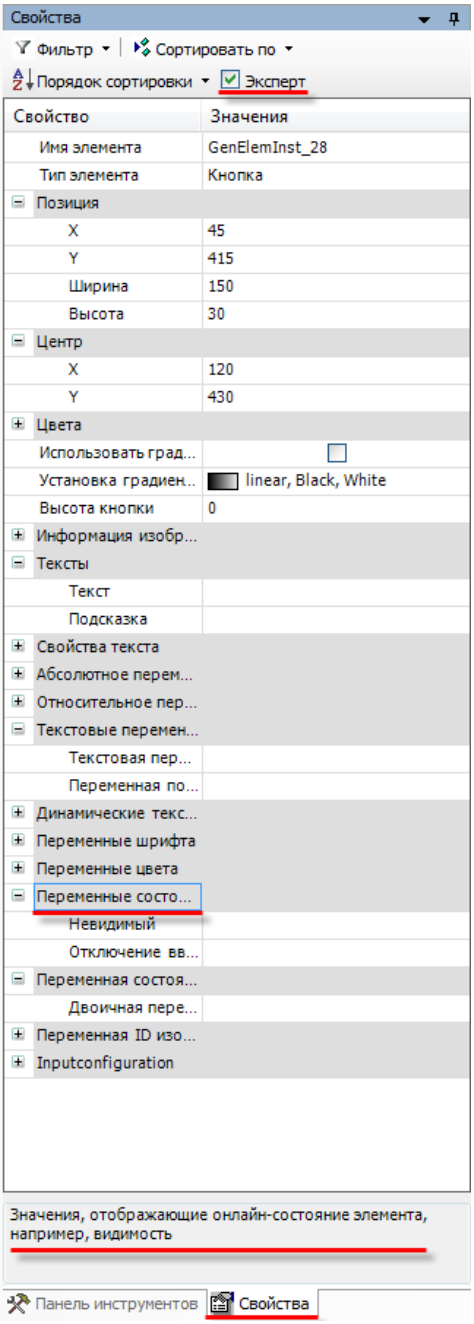


Рис. 7.11. Панель свойств кнопки

Панель свойств состоит из раскрывающихся вкладок с параметрами, часть которых являются идентичными для всех элементов, часть – уникальными. Для отображения всех вкладок необходимо использовать режим **Эксперт**, который включается вверху панели. При выделении любой из вкладок внизу отображается информационная подсказка. Для изменения параметра достаточно два раза нажать **ЛКМ** на его значение.

В дальнейшем мы будем рассматривать только те настройки элементов, которые понадобятся нам при создании проекта. Подробное описание всех настроек приведено в справке CODESYS.

Настроим кнопку следующим образом (изменения в стандартных настройках выделены красным):

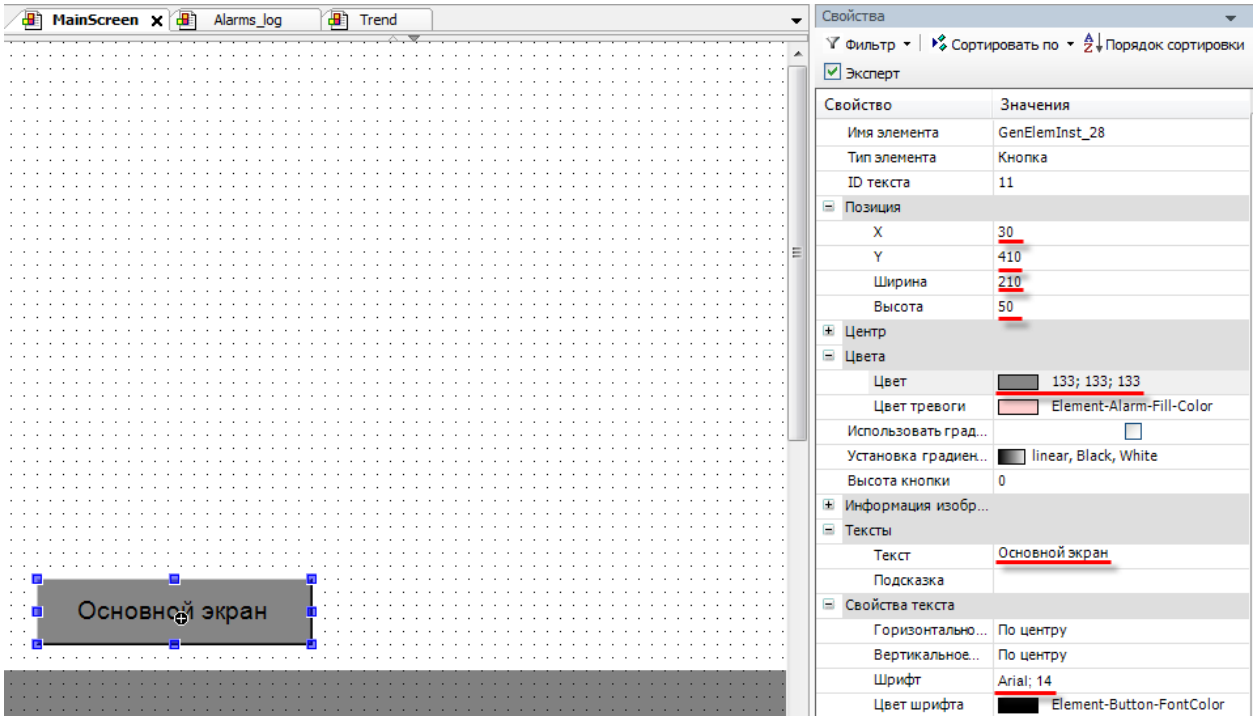


Рис. 7.12. Настройка кнопки перехода

Для того, чтобы изменить размер элемента, его положение на экране и текст, необязательно менять настройки через **Панель свойств** – все эти операции можно проделать непосредственно нажимая **ЛКМ** на элемент:

1. Для **перетаскивания** выделите элемент, наведите на него мышью для появления курсора перетаскивания () и, зажав **ЛКМ**, перетащите элемент в нужное место экрана;

2. Для **изменения размера** выделите элемент, наведите мышью на одну из его опорных точек () для появления курсора деформации () и, зажав **ЛКМ**, растяните элемент до нужных размеров;

3. Для **изменения текста элемента** наведите на текстовое поле элемента мышью для появления курсора ввода (), после чего однократным нажатием **ЛКМ** перейдите к редактированию текста.

Созданный элемент можно копировать/вставлять с помощью **контекстного меню** (открывается по нажатию **ПКМ** на элемент), либо с помощью стандартных комбинаций клавиш Ctrl+C/Ctrl+V.

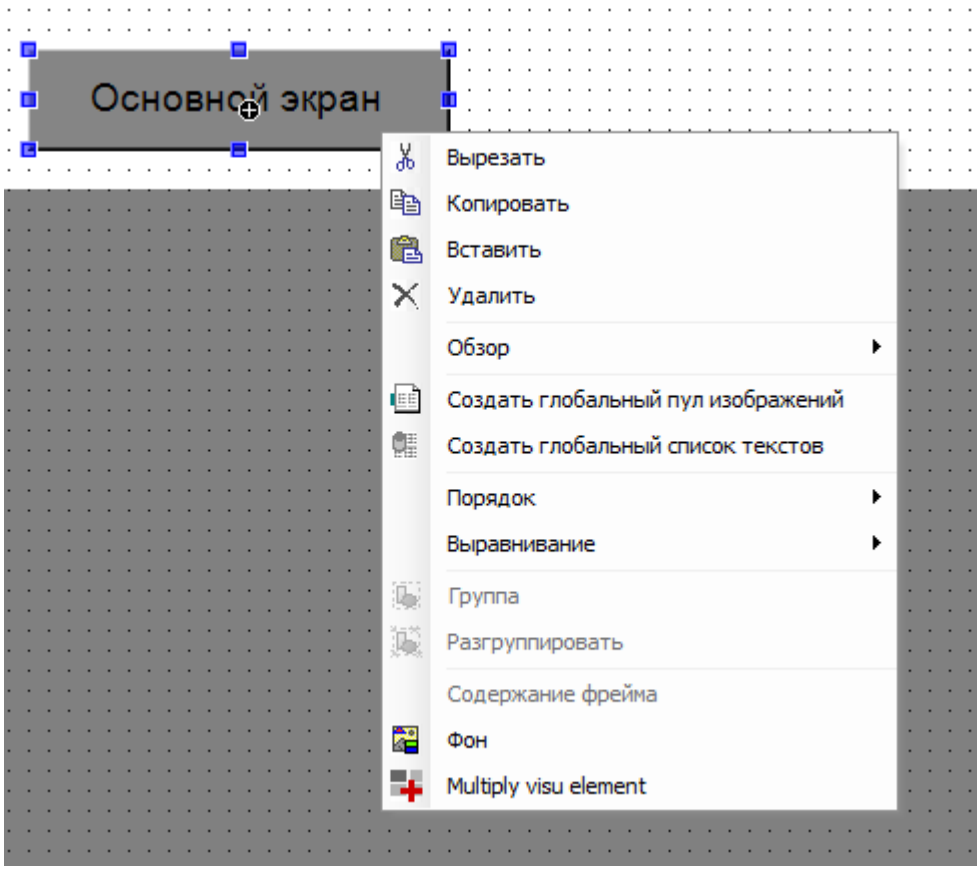


Рис. 7.13. Контекстное меню элемента

На основе созданной кнопки создадим еще две (красным выделены настройки, значения которых отличаются от настроек кнопки **Основной экран**):

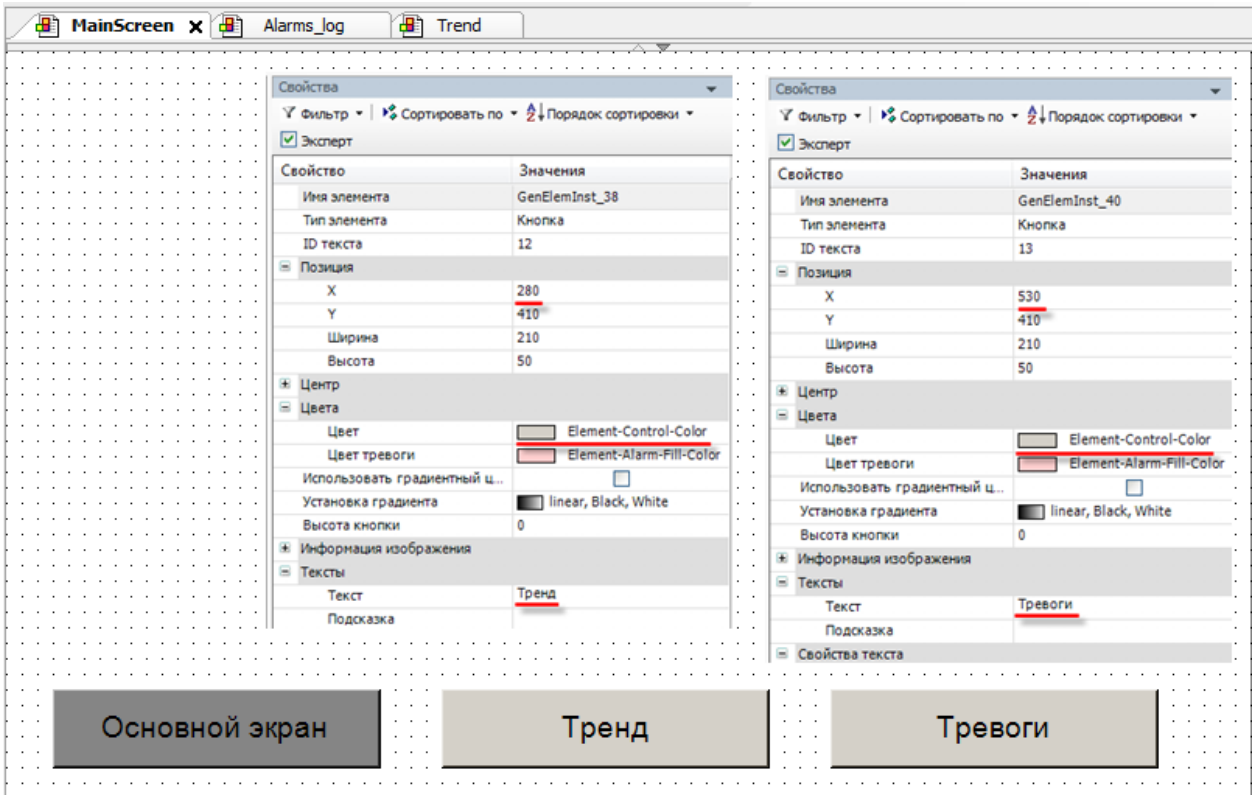


Рис. 7.14. Кнопки перехода

Темно-серый цвет кнопки **Основной экран** характеризует отображение этого экрана в данный момент времени; соответственно, на **экране Trend** темно-серой будет **кнопка Тренд**, а цвета кнопок **Тревоги** и **Основной экран** будут совпадать.

Зажав **ЛКМ**, выделим сразу три кнопки и с помощью команд контекстного меню **Копировать**/**Вставить**, перенесем их на экраны **Trend** и **Alarms_log**:

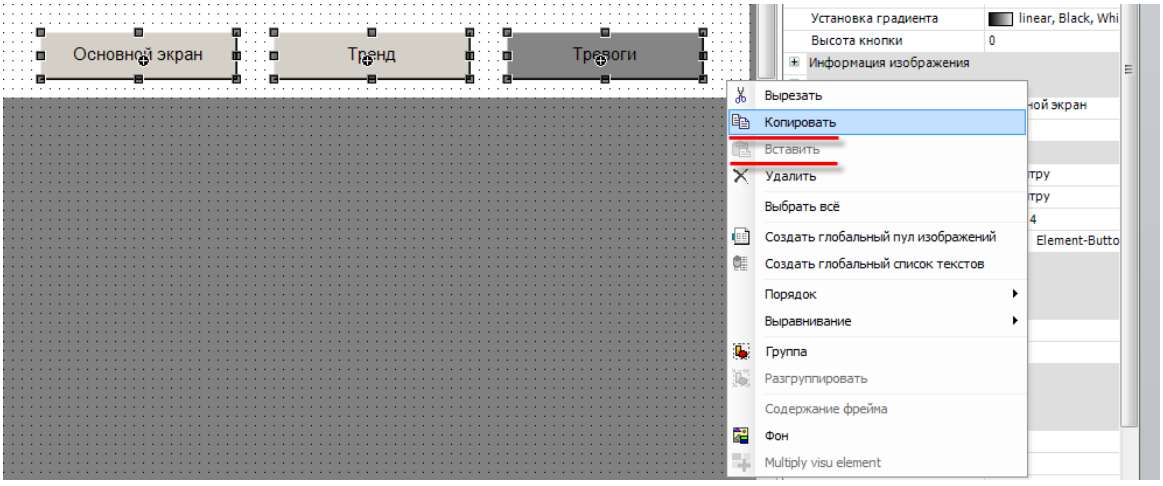


Рис. 7.15. Копирование группы элементов

Рис. 7.16. Внешний вид кнопок на экране **Trend**

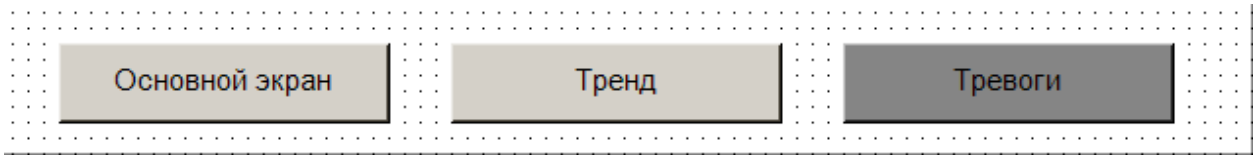


Рис. 7.17. Внешний вид кнопок на экране **Alarms_log**

Итак, мы создали кнопки переключения между экранами. ***Их функциональная настройка будет описана*** ***п. 7.5*****.**

Элемент **Кнопка** также подходит для создания фона заголовков, информационных панелей и т.д. ***В нашем примере мы будем создавать панели с помощью кнопок; пользователь может использовать для панелей и другие элементы, например, Прямоугольник.***

Создадим панель для отображения системного времени контроллера:

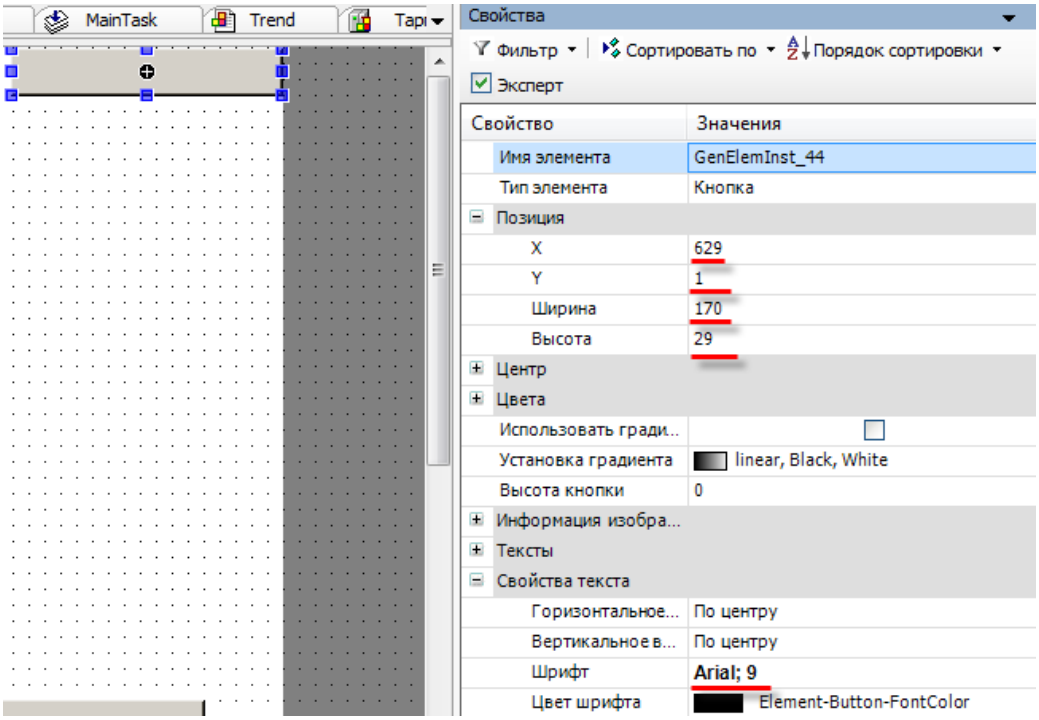


Рис. 7.18. Настройка панели отображения системного времени

Функциональная настройка панели системного времени будет описана в ***п. 7.5****; разместим эту панель на всех экранах проекта.*

Создадим еще четыре панели – панель названия экрана, панель названия устройства управления (кондиционера), панель управления температурой и панель кондиционера. От созданных ранее они отличаются только размерами, координатами и надписями:

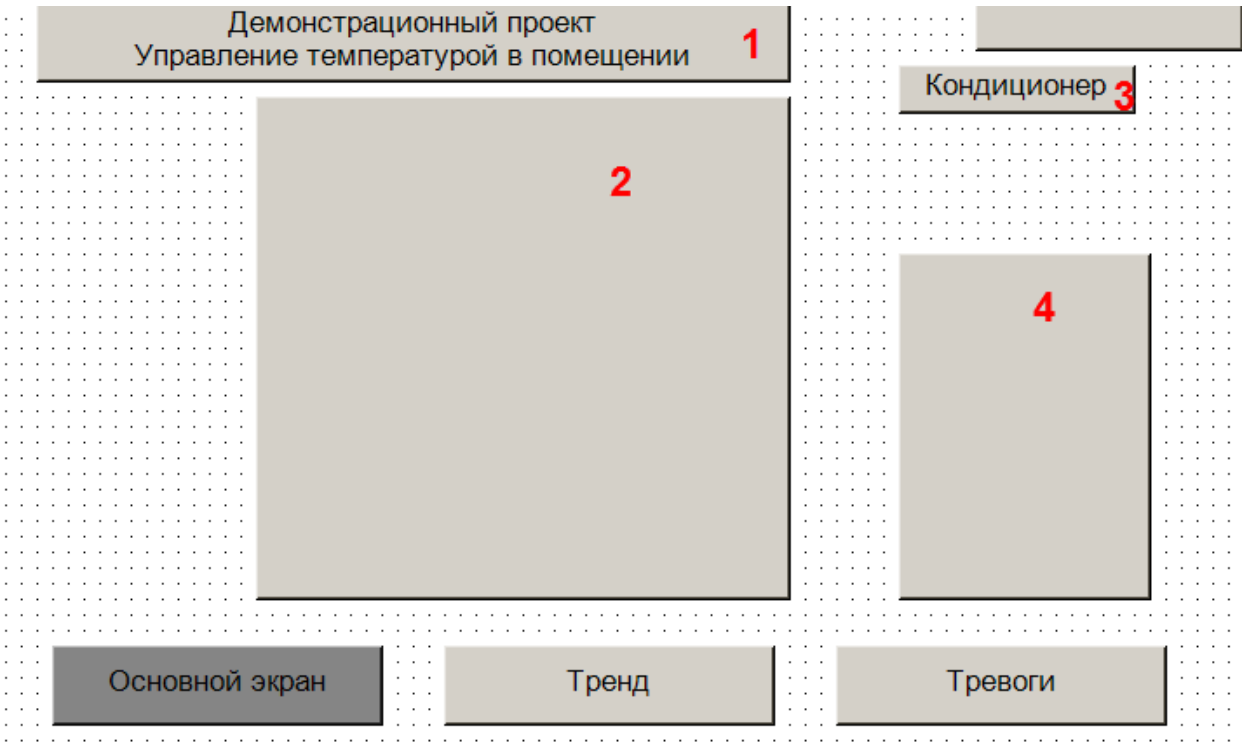


Рис. 7.19. Экран **MainScreen** после окончания размещения элементов Кнопка

Настройки панелей следующие:

Номер панели на рис. 7.19	**Текст**	**Настройки**			
		X	**Y**	**Ширина**	**Высота**
1	Демонстрационный проект Управление температурой в помещении	30	0	480	50
2	-	170	60	340	320
3	Кондиционер	580	40	150	30
4	-	580	160	160	220

Для перехода на следующую строку при наборе текста (панель 1) используется комбинация клавиш **Ctrl+Enter**.

II. Элемент Текстовое поле

Элемент **Кнопка**, который мы рассматривали в предыдущем пункте, уже содержит возможность ввода текста; но достаточно часто приходится размещать на панелях значительное количество надписей с разными настройками (размер, шрифт и т.д.). Для этих целей удобно использовать элемент **Тестовое поле**, расположенный во **вкладке Стандартные элементы управления**:

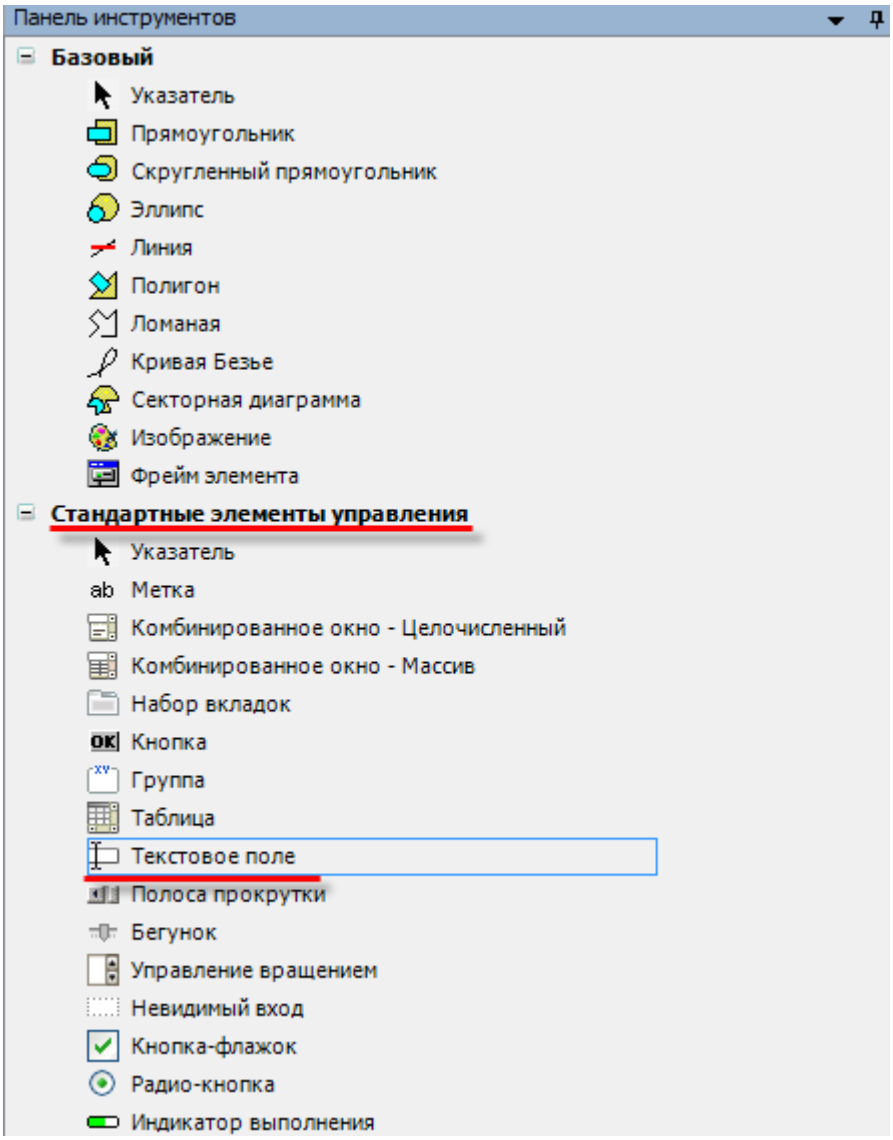


Рис. 7.20. Элемент **Текстовое поле**

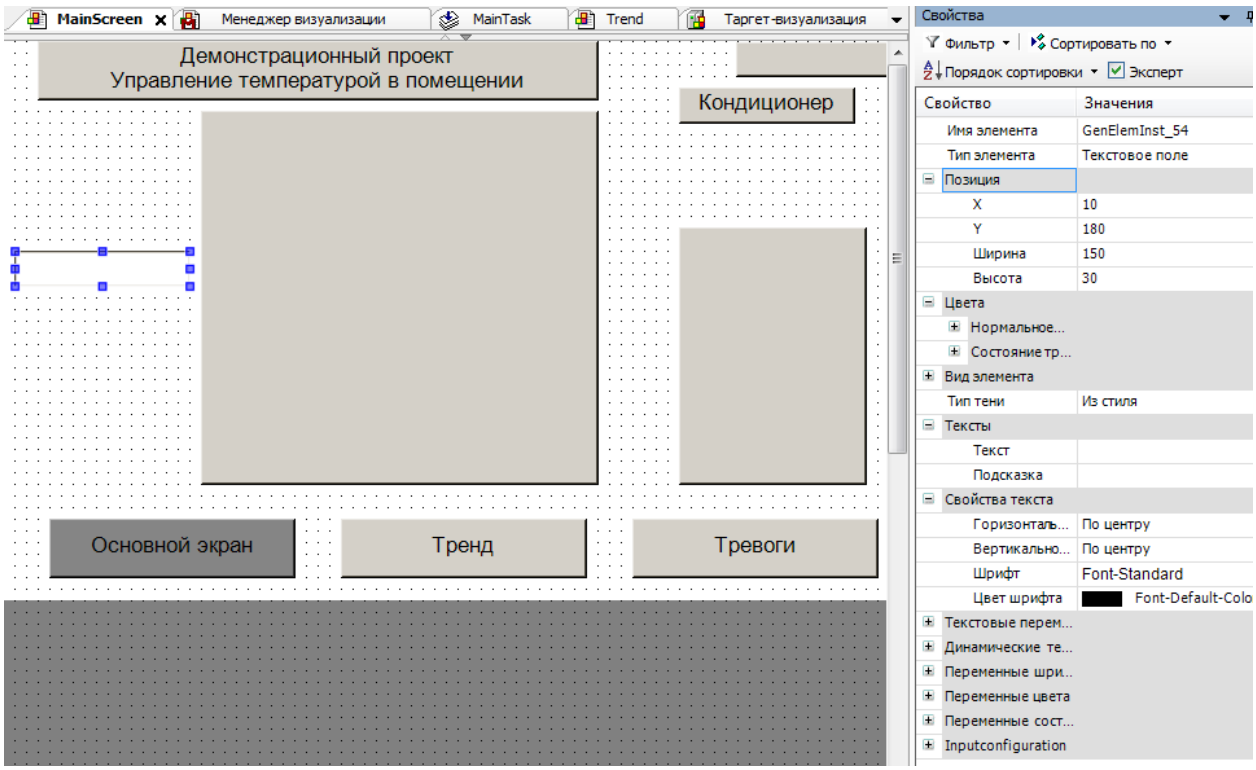


Рис. 7.21. Размещение элемента **Текстовое поле** в рабочей области и его настройки

Настройка элемента совершенно аналогична настройке элемента **Кнопка**.

Разместим на панели управления температурой ее заголовок:

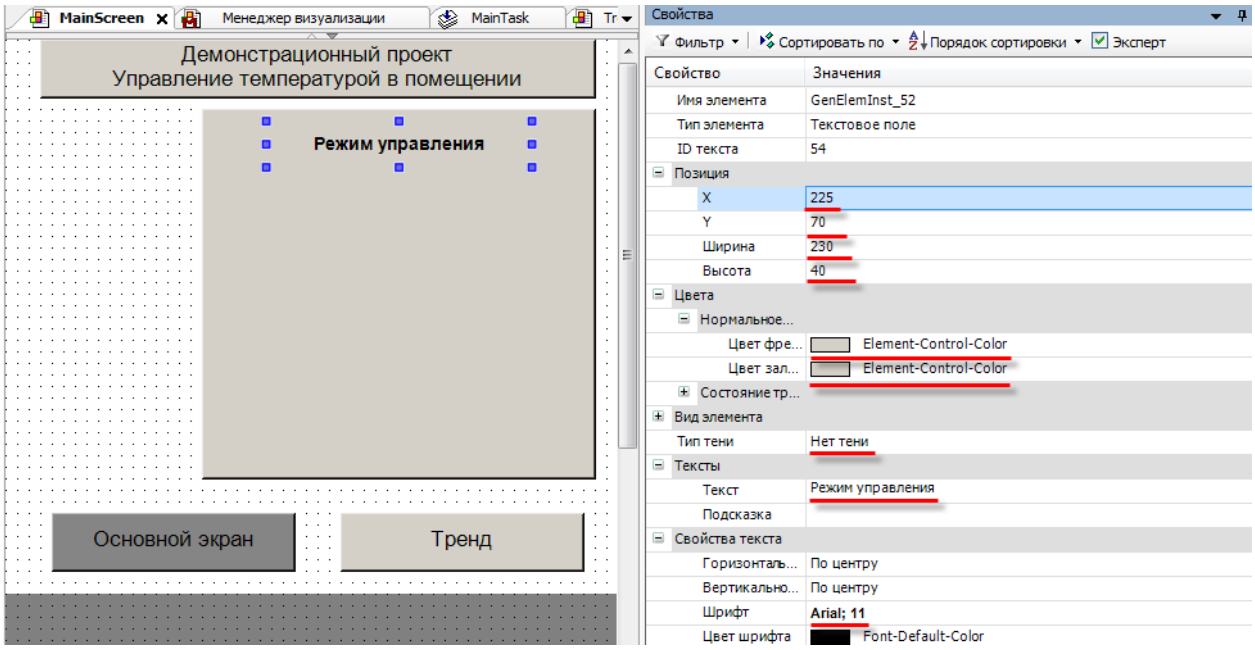


Рис. 7.22. Настройки заголовка панели управления температурой

Разместим еще четыре текстовых поля:

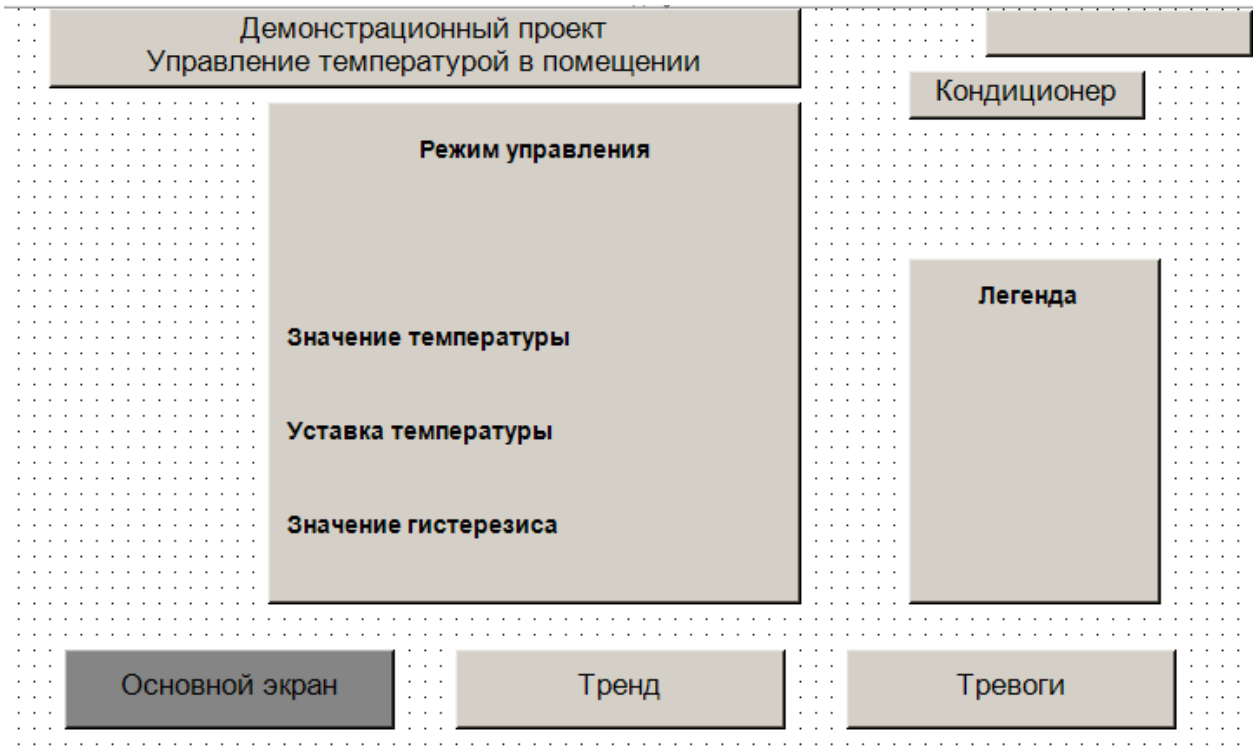


Рис. 7.23. Экран **MainScreen** после окончания размещения элементов **Тестовое поле**

Настройки текстовых полей следующие:

Текст	**Горизонтальное выравнивание (вкладка Свойства текста)**	**Настройки**			
		X	**Y**	**Ширина**	**Высота**
Значение температуры	лево	180	190	223	40
Уставка температуры	лево	180	250	223	40
Значение гистерезиса	лево	180	310	223	40
Легенда	по центру	610	169	90	30

Для операций с несколькими элементами нужно, зажав **ЛКМ**, выделить область рабочего поля (или последовательно выделять элементы при зажатой клавише **shift**), после чего нажать **ПКМ** на любом месте рабочего поля для вызова контекстного меню:

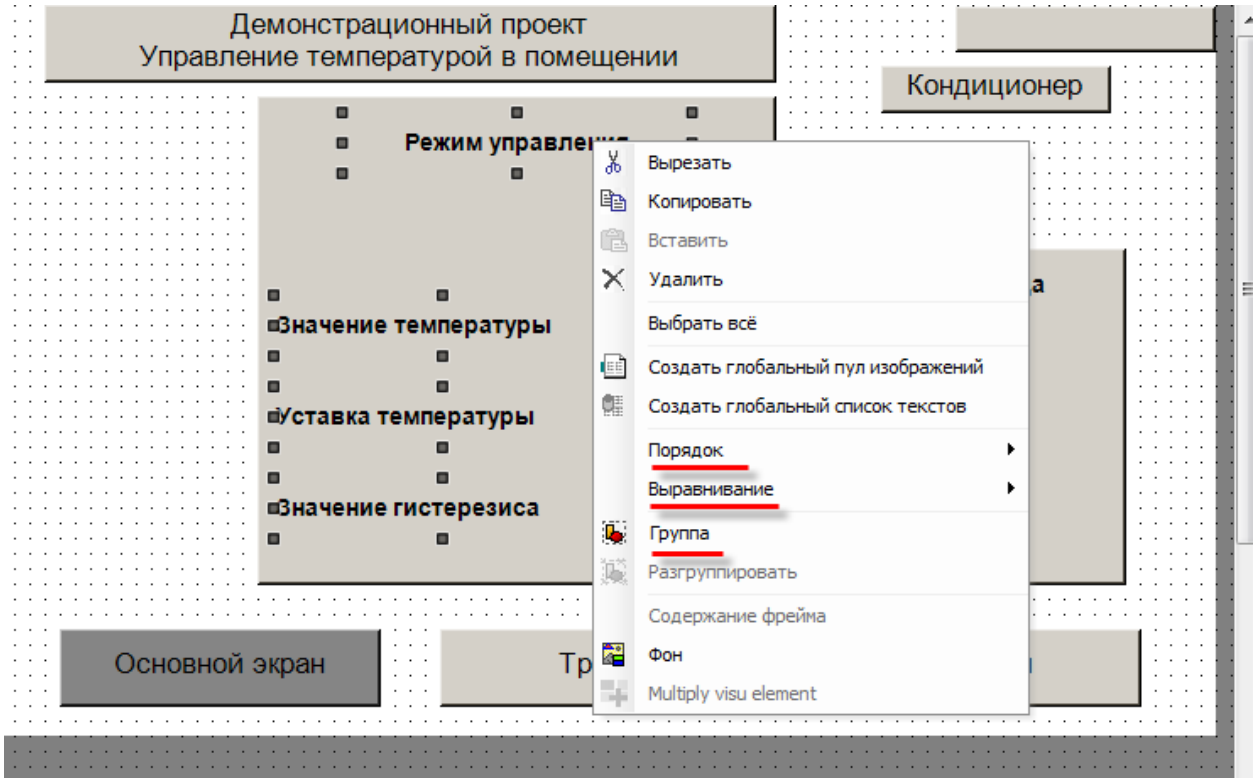


Рис. 7.24. Контекстное меню группы элементов

Вкладка **Выравнивание** позволяет выравнивать элементы относительно друг друга; при этом выравнивание происходит ***относительно элемента, выделенного последним***.

Вкладка **Группа** позволяет сгруппировать несколько элементов и взаимодействовать с ними как с одним элементом.

Вкладка **Порядок** (доступная и при выделении одного элемента) позволяет размещать элементы в разных **слоях** относительно друг друга; это позволяет накладывать элементы друг на друга без перекрывания.

III. Элемент Кнопка-флажок (чекбокс)

Кнопки-флажки (чекбоксы) позволяют пользователю управлять параметром с двумя состояниями – одно из них соответствует пустому чекбоксу, второе – заполненному (традиционно используются пиктограмма «галочка»).

В нашем примере мы будем использовать чекбокс для переключения режимов управления температурой – **автоматического** (в этом случае значение температуры в помещении измеряется датчиком, а управляющий сигнал передается на реальное устройство) и **эмуляции** (показания датчика задаются пользователем, вместо выдачи сигнала управления осуществляется **эмуляция** процесса регулирования).

Разместим **чекбокс** на экране **MainScreen**, под надписью **Режим управления**:

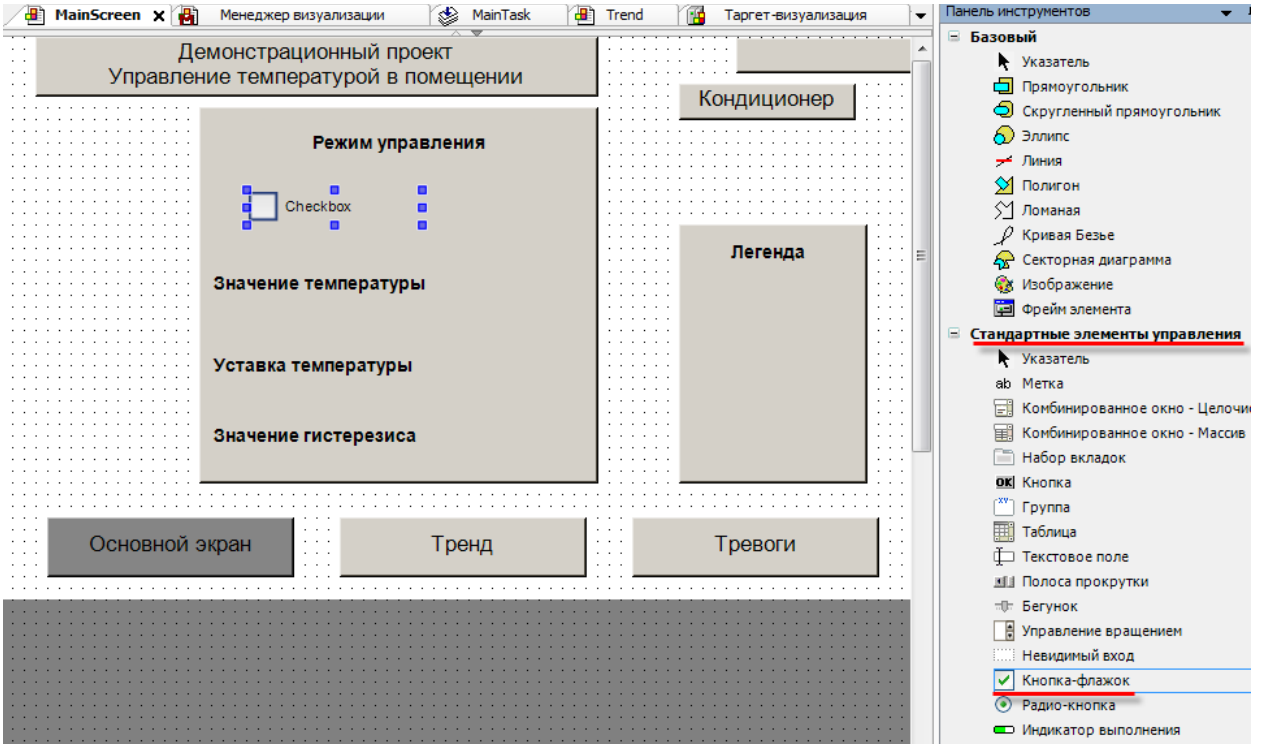


Рис. 7.25. Добавление элемента **Чекбокс**

Настроим его следующим образом:

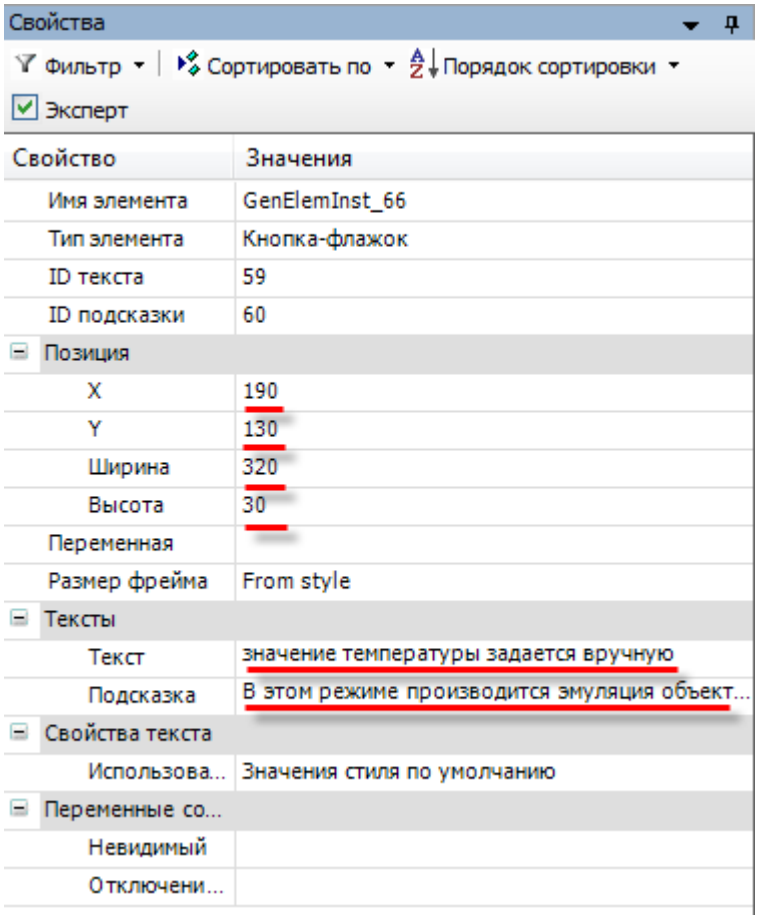


Рис. 7.26. Настройка элемента **Чекбокс**

Подсказка представляет собой текст, всплывающий при наведении на элемент курсора. ***Подсказка видна только при запуске проекта.***

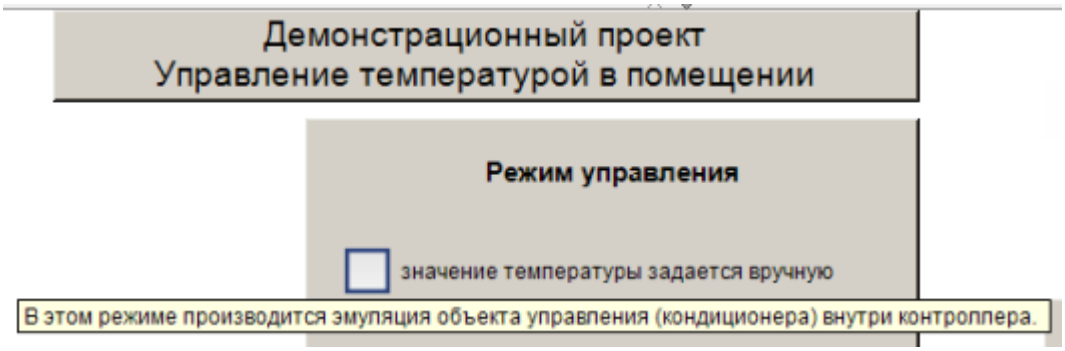
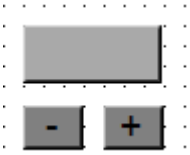


Рис. 7.27. Всплывающая подсказка

IV. Элемент Кнопка (продолжение пп. I)

Теперь добавим на панель управления температурой информационные окна для отображения значений параметров и кнопки управления этими параметрами. Создадим их с помощью элемента **Кнопка**:



Процесс создания кнопок описан в пп. I; поскольку шрифт, цвет и т.д. не имеют принципиального значения, то подробные настройки элементов мы в этот раз не приводим.

Для того, чтобы разместить эти кнопки поверх панели управления температурой, перенесем их в ***слой выше***:

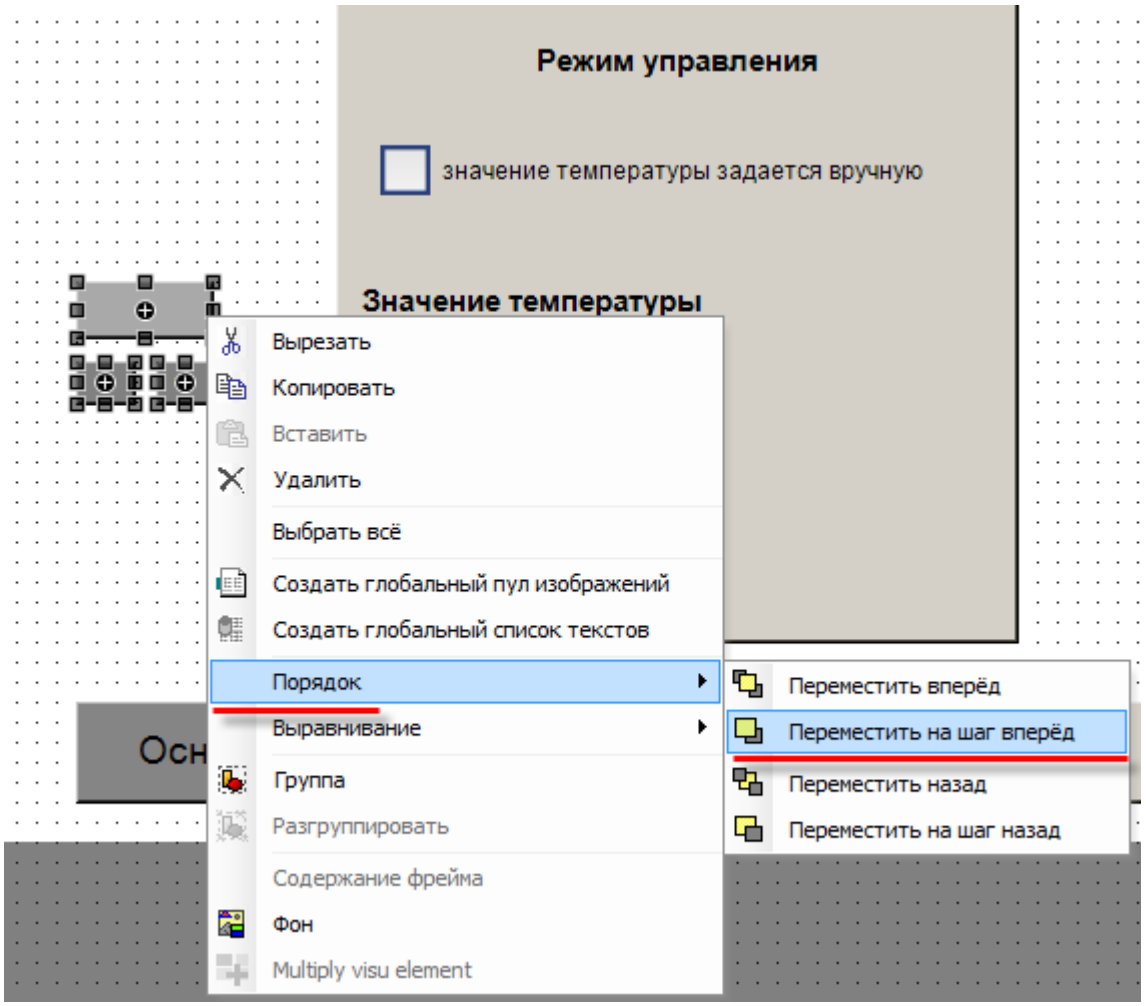


Рис. 7.28. Перенос элементов на слой выше

Скопируем созданные нами элементы и разместим их на панели управления температурой:

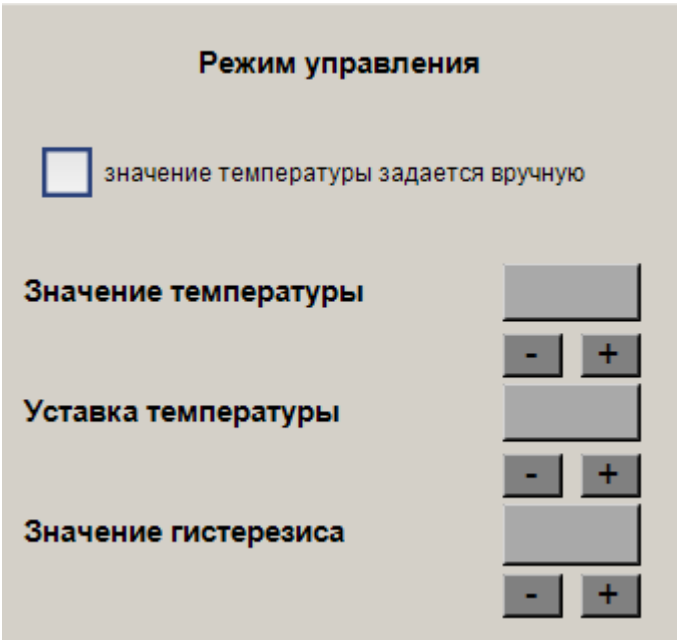


Рис. 7.29. Панель управления температурой – окончательный вариант

V. Элемент Отображение линейки

Для отображения текущего значения температуры в помещении воспользуемся элементом **Отображение линейки** (вкладка **Элементы управления измерением**).

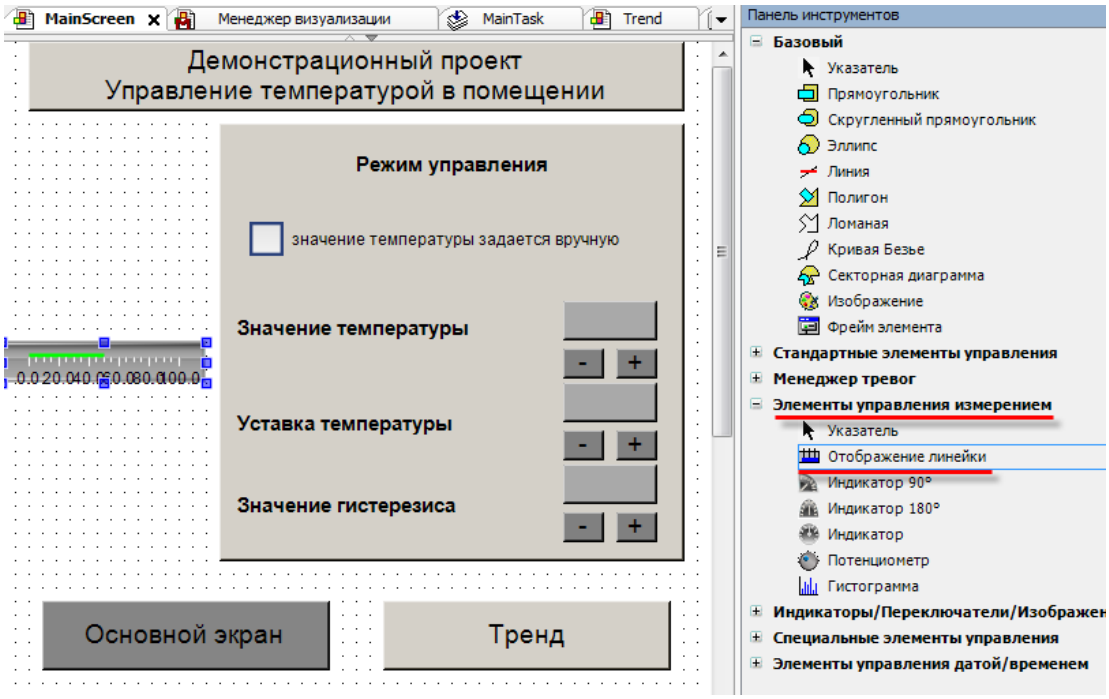


Рис. 7.30. Добавление элемента **Отображение линейки**

Настроим элемент следующим образом:

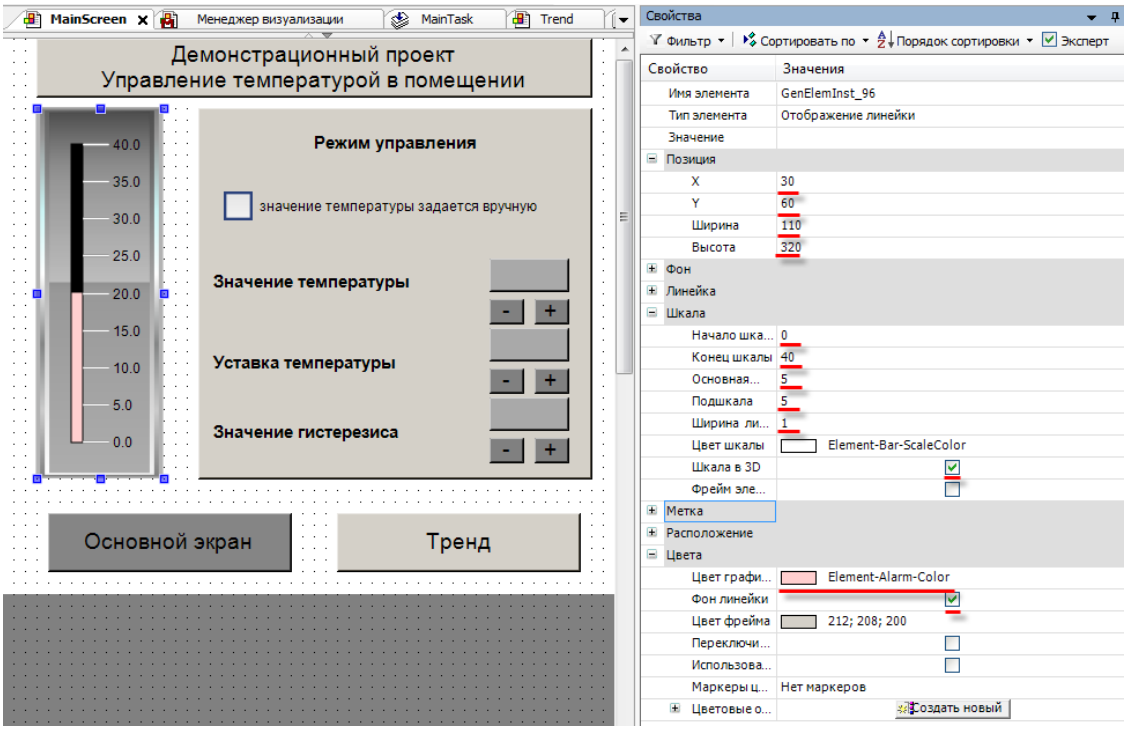


Рис. 7.31. Настройки элемента **Отображение линейки**

VI. Элемент Клавишный выключатель

Для включения/выключения кондиционера воспользуемся элементом **Клавишный выключатель** (вкладка **Индикаторы/Переключатели/Изображения**).

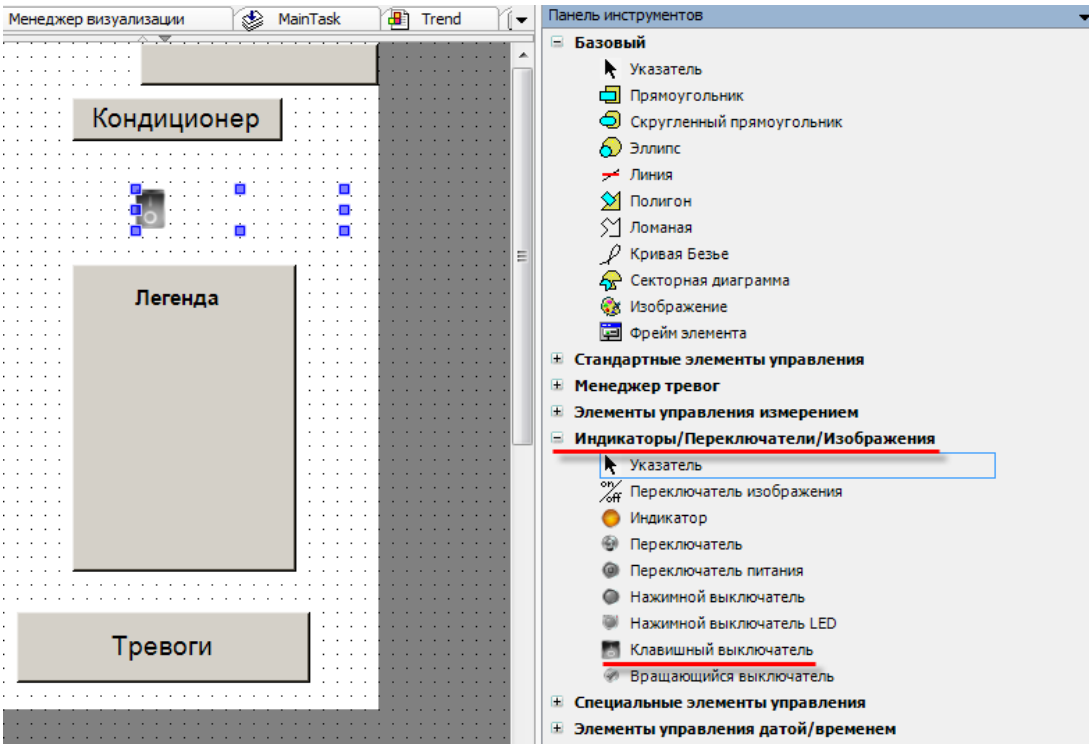


Рис. 7.32. Добавление элемента **Клавишный выключатель**

Для клавишного выключателя настроим только координаты и размеры:

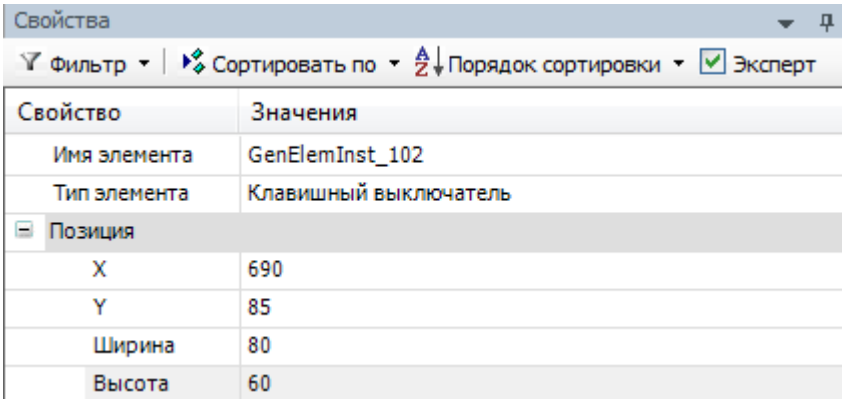


Рис. 7.33. Настройки элемента **Клавишный выключатель**

VII. Элемент Индикатор

Для отображения текущего режима работы кондиционера воспользуемся элементом **Индикатор** (вкладка **Индикаторы/Переключатели/Изображения**).

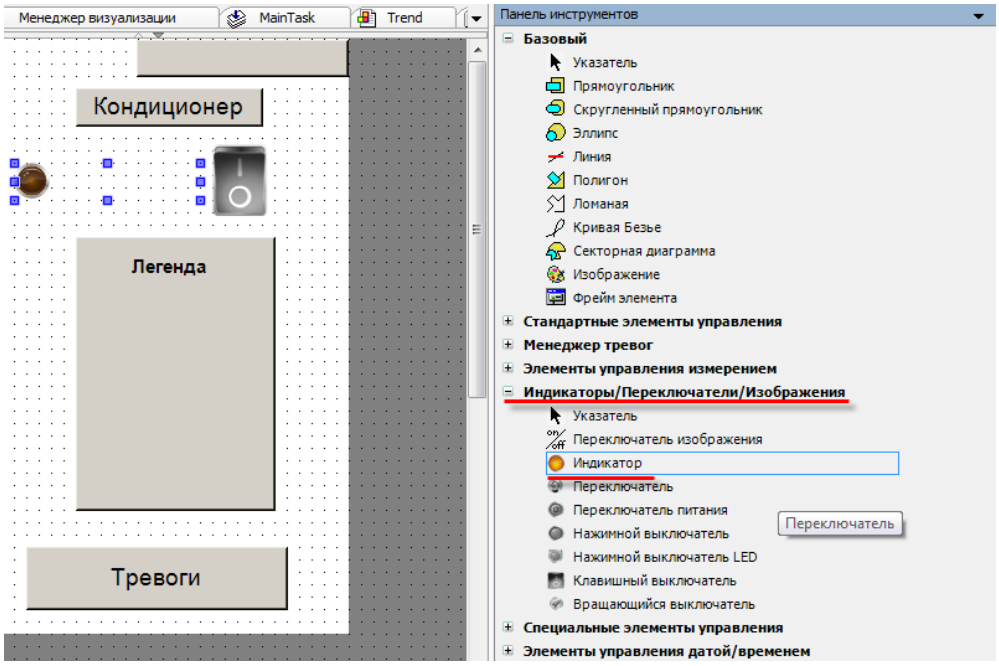


Рис. 7.34. Добавление элемента **Индикатор**

Кондиционер имеет **четыре режима работы** – нагрев, охлаждение, режим ожидания, выключен. Реализовать отображение четырех состояний с помощью одного графического элемента не представляется возможным, поэтому подготовим четыре индикатора разных цветов; после привязки к ним переменных (п. 7.5), мы наложим эти индикаторы друг на друга для создания эффекта **многопозиционного индикатора**.

Пока что поместим эти индикаторы ***за пределами рабочей области***; все они будут обладать одинаковыми размерами (ширина – 80, высота – 70) и разными цветами:

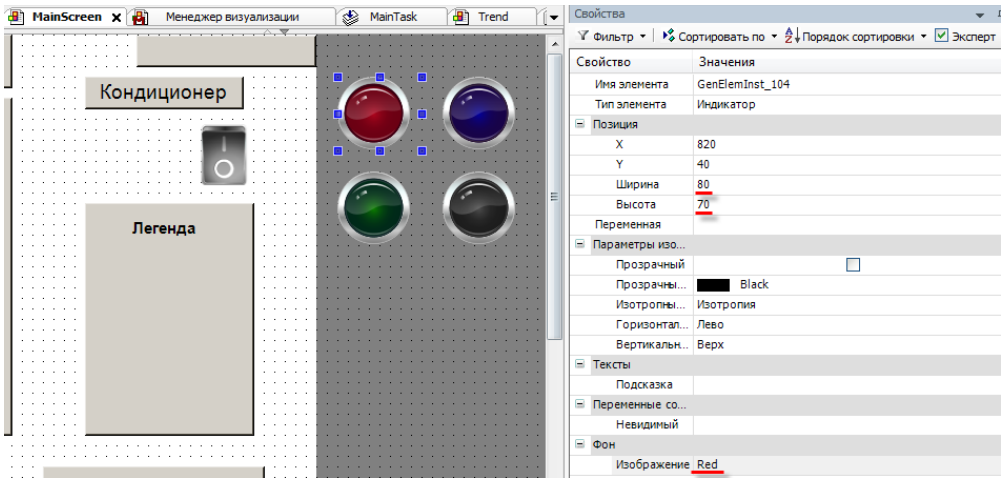


Рис. 7.35. Настройки элемента **Индикатор**

Такие же индикаторы добавим на панель легенды кондиционера и снабдим их поясняющими надписями (создание надписей описано в пп. II):

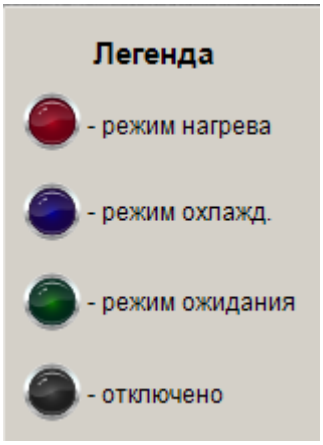


Рис. 7.36. Панель легенды кондиционера – окончательный вариант

Итак, после выполнения действий, описанных в п. 7.3.2, экран визуализации **MainScreen** выглядит следующим образом:

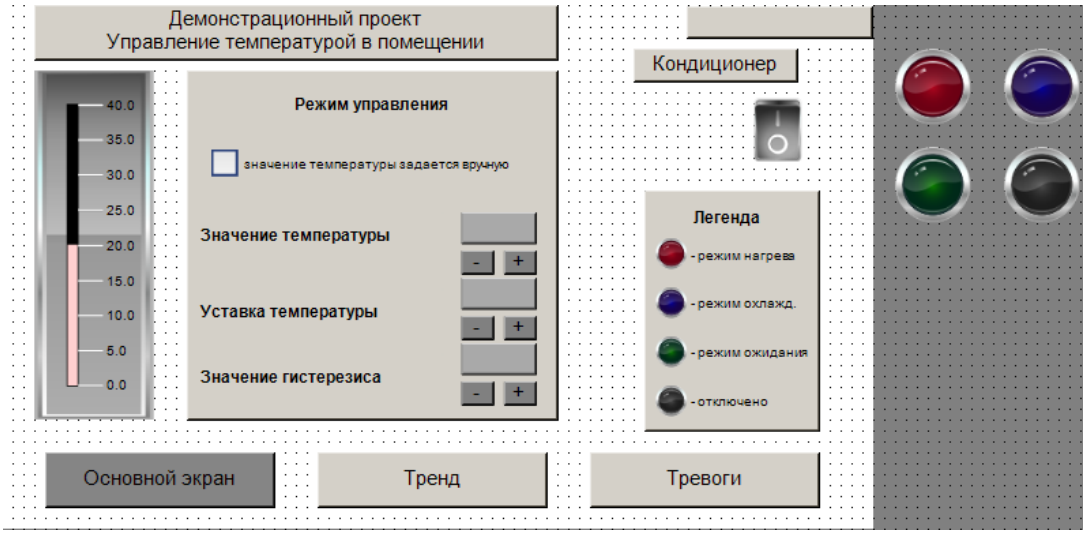


Рис. 7.37. Внешний вид экрана **MainScreen** (после п. 7.3.2)

Окончательная доработка экрана (привязка переменных к визуализации, настройка действий кнопок, наложение индикаторов друг на друга) будет описана в п. 7.5.

7.3.3. Наполнение экрана Trend

Экран **Trend** будет использоваться для графического отображения текущего значения параметров (температуры, уставов температуры, уставок гистерезиса, аварийных уставок), установки значений аварийных уставок и сигнализации о выходе температуры за аварийные уставки с помощью мигания аварийной лампы.

После выполнения действий, описанных в п. 7.3.2 пп. I (добавление кнопок перехода между экранами и панели системного времени), экран **Trend** выглядит следующим образом:

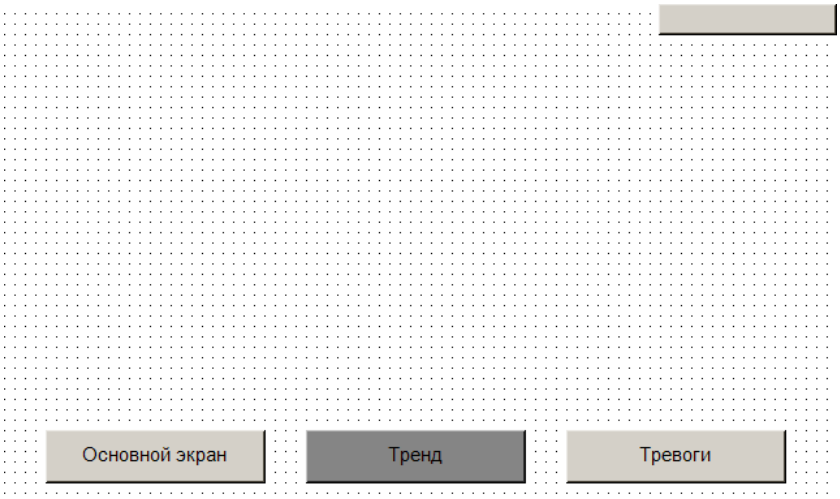


Рис. 7.38. Внешний вид экрана **Trend** (после п. 7.3.2)

Добавим на него название экрана (элемент **Кнопка**), панель аварийных уставок (элемент **Кнопка** с наложенными поверх элементами **Текстовое поле**) и аварийную лампу (элемент **Индикатор**) с пустым названием (элемент **Кнопка**). Название лампы будет задаваться пользователем в процессе работы проекта.

Процесс добавления и настройки этих элементов описан в п. 7.3.2, пп. I, пп. IV (**Кнопка**), пп. II (**Текстовое поле**), пп. VII (**Индикатор**). В результате экран должен выглядеть следующим образом:

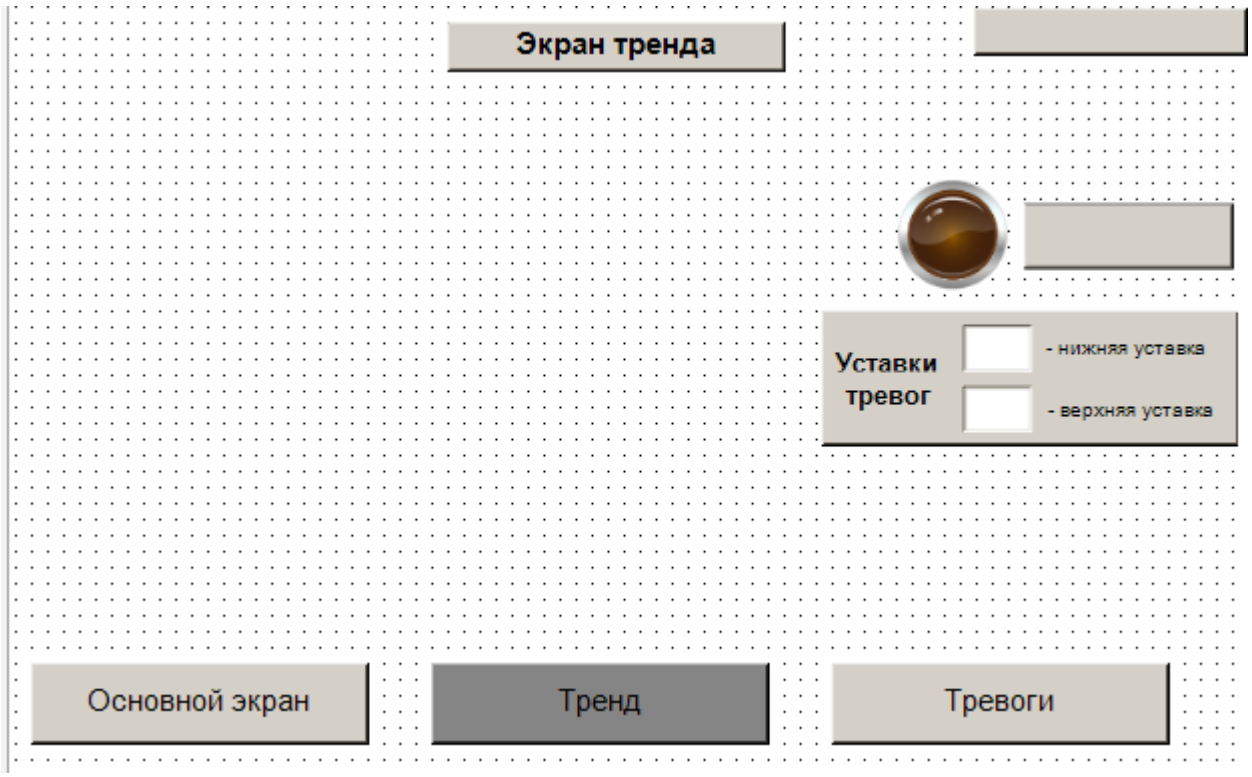


Рис. 7.39. Внешний вид экрана **Trend**

Для графического отображения значений переменных в **CODESYS** существует два элемента – **Трассировка** и **Тренд**. Трассировка отображает значения только за крайне ограниченный отрезок времени, в то время как **Тренд** позволяет просматривать историю изменений переменных. Настройка элементов практически аналогична; ***в данном примере мы будем использовать только элемент Тренд.***

Добавим элемент **Тренд** на экран **Trend** (вкладка **Специальные элементы управления**). Появится окно конфигурации тренда. Не выполняя никаких настроек (они будут рассмотрены в п. 7.5), нажмем **ОК**:

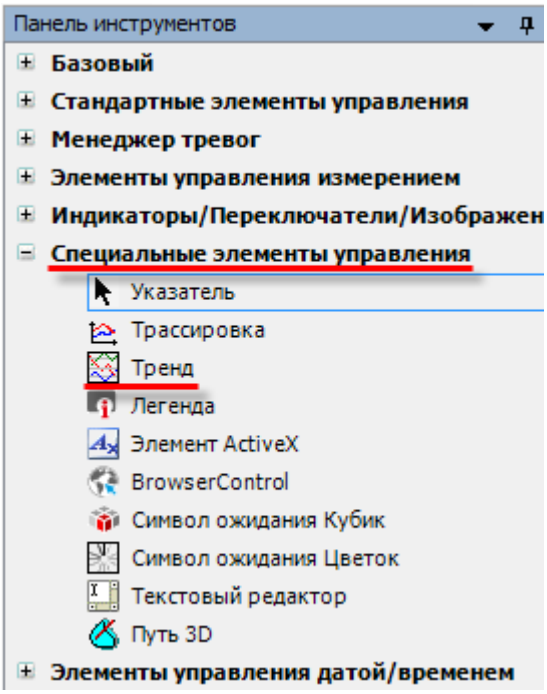


Рис. 7.40. Элемент **Тренд** на **Панели инструментов**

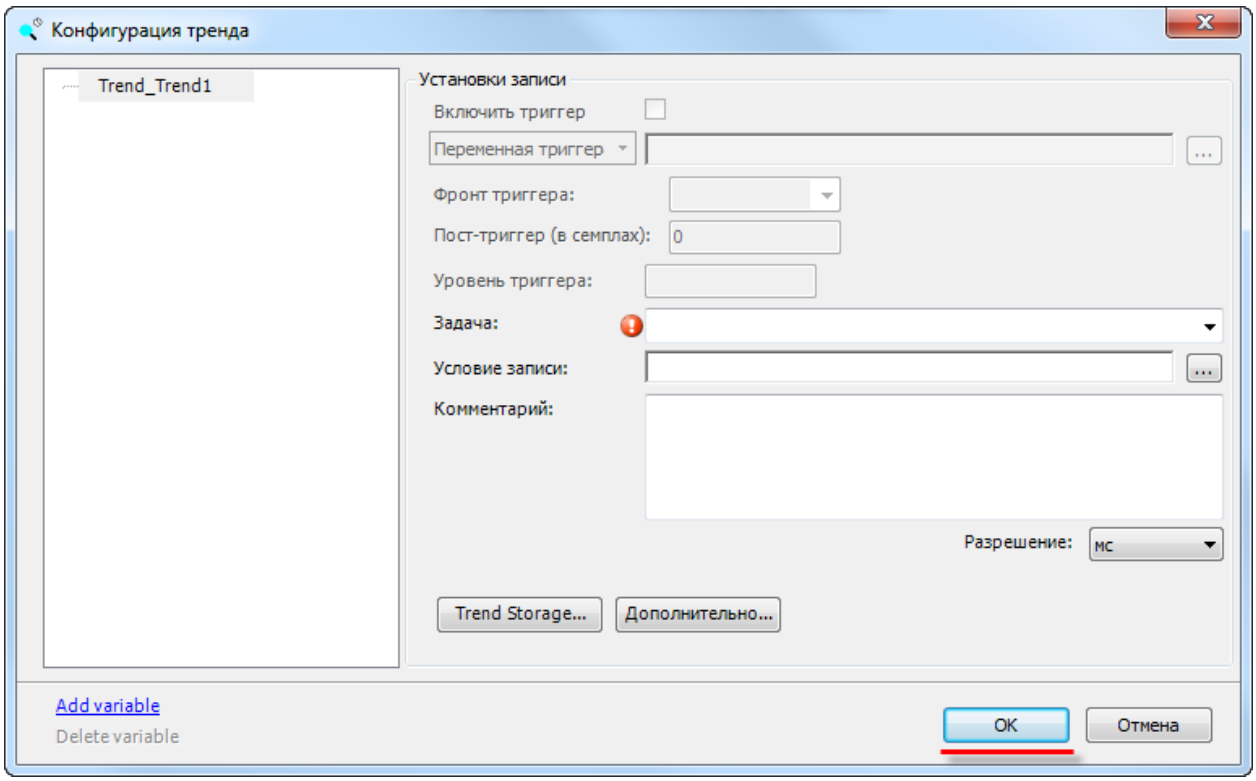


Рис. 7.41. Окно конфигурации тренда

Обратим внимание, что после добавления элемента **Тренд** на **Панели устройств** появились новые компоненты:

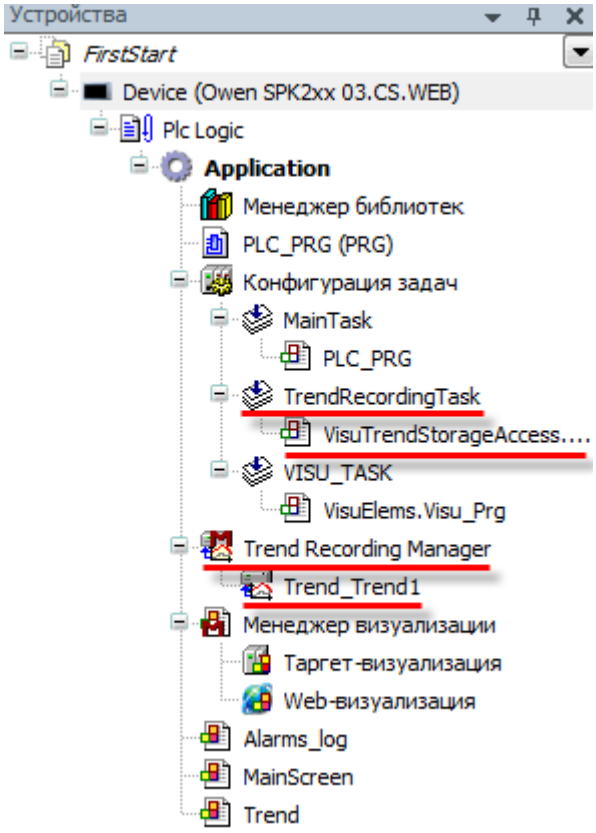


Рис. 7.42. Панель инструментов после добавления элемента **Тренд**

Настроим тренд следующим образом:

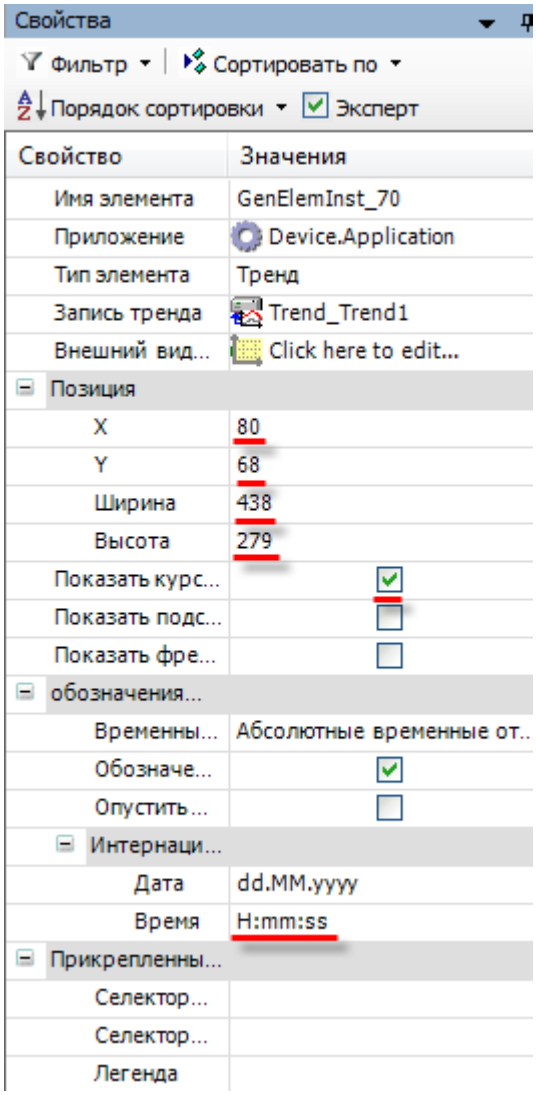


Рис. 7.43. Настройка элемента **Тренд**

После настройки тренд будет выглядеть следующим образом:

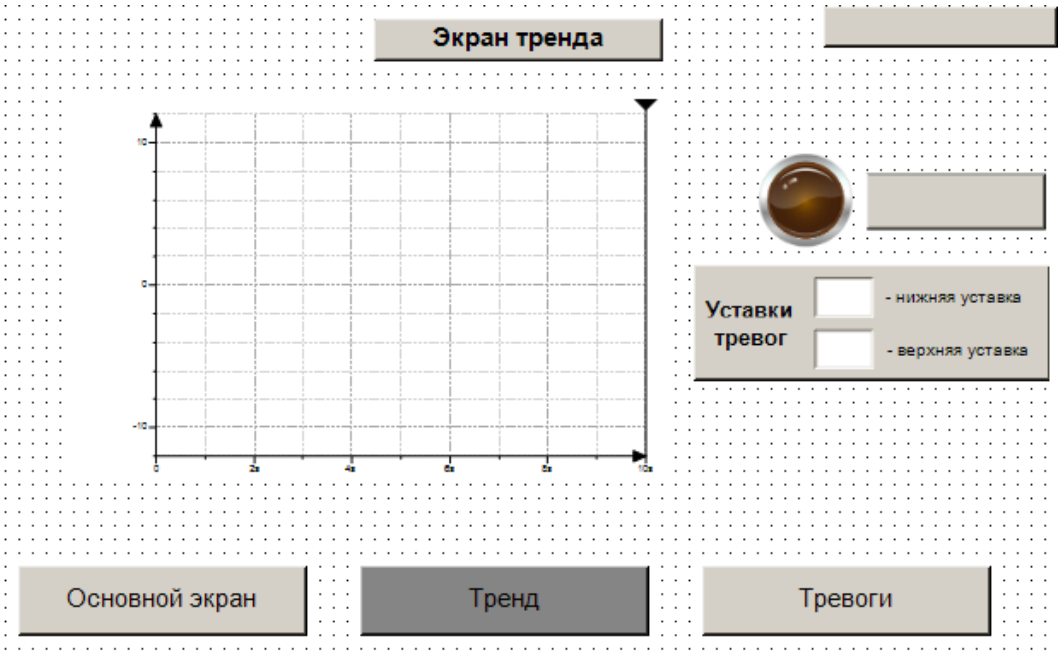


Рис. 7.44. Экран **Trend** после добавления тренда

Нажмем на тренд **ПКМ** и выберем пункт **Вставить компоненты для управления трендом**:

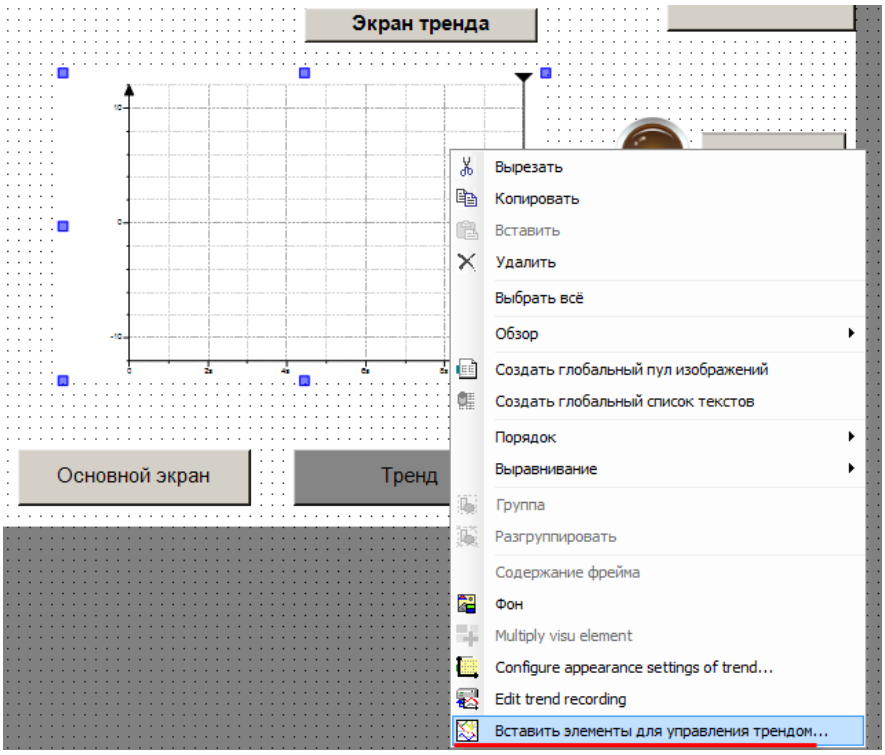


Рис. 7.45. Добавление элементов управления трендом

Настройки **Мастера трендов** оставим по умолчанию:

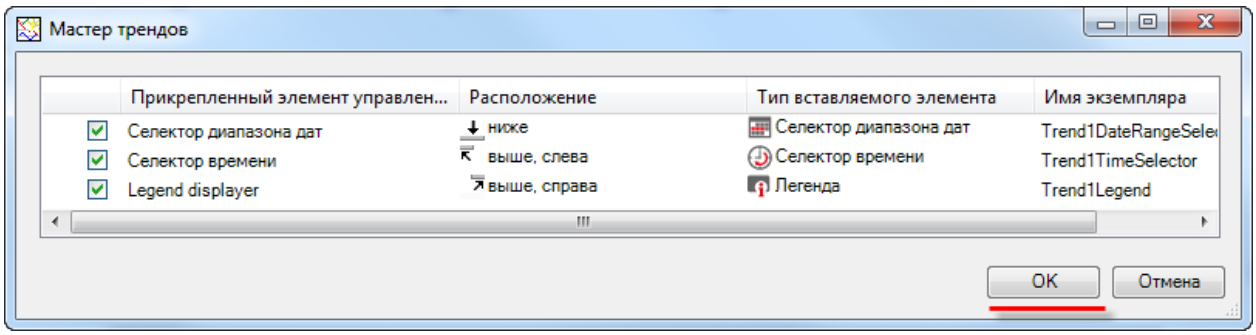


Рис. 7.46. Меню **Мастера трендов**

В результате экран будет выглядеть так:

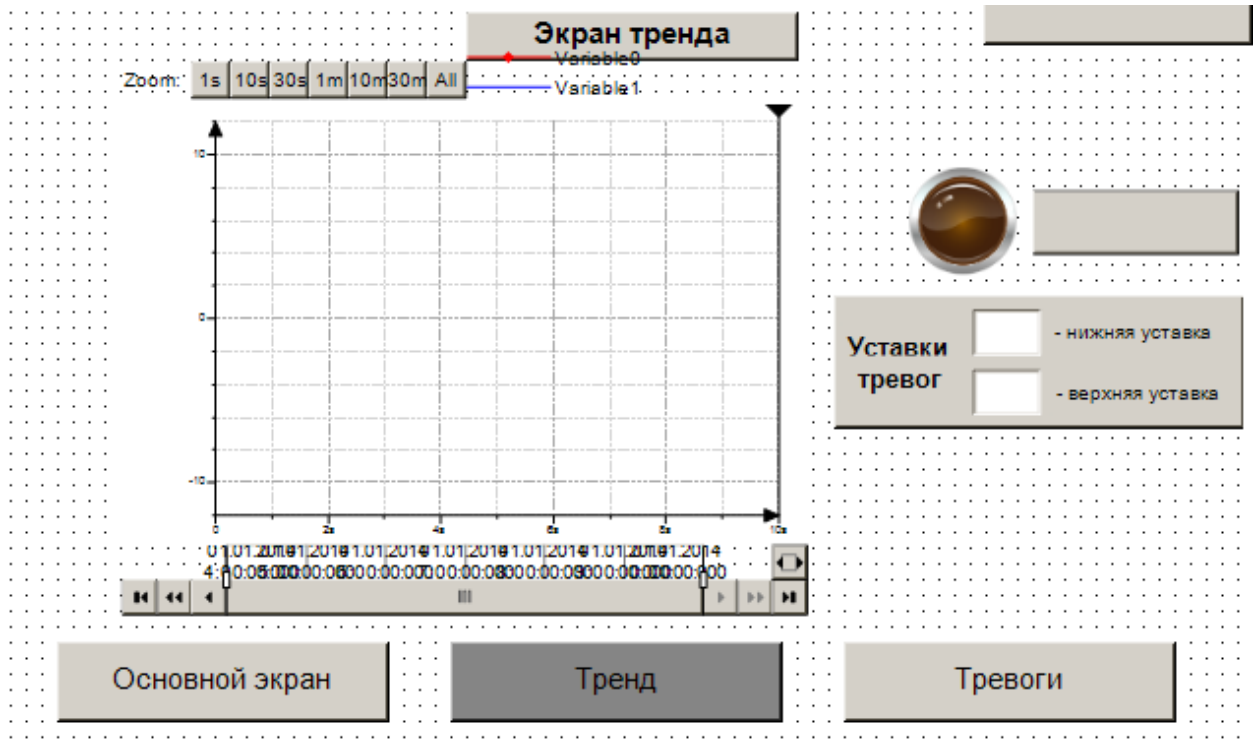


Рис. 7.47. Элементы управления трендом

Настроим компоненты следующим образом:

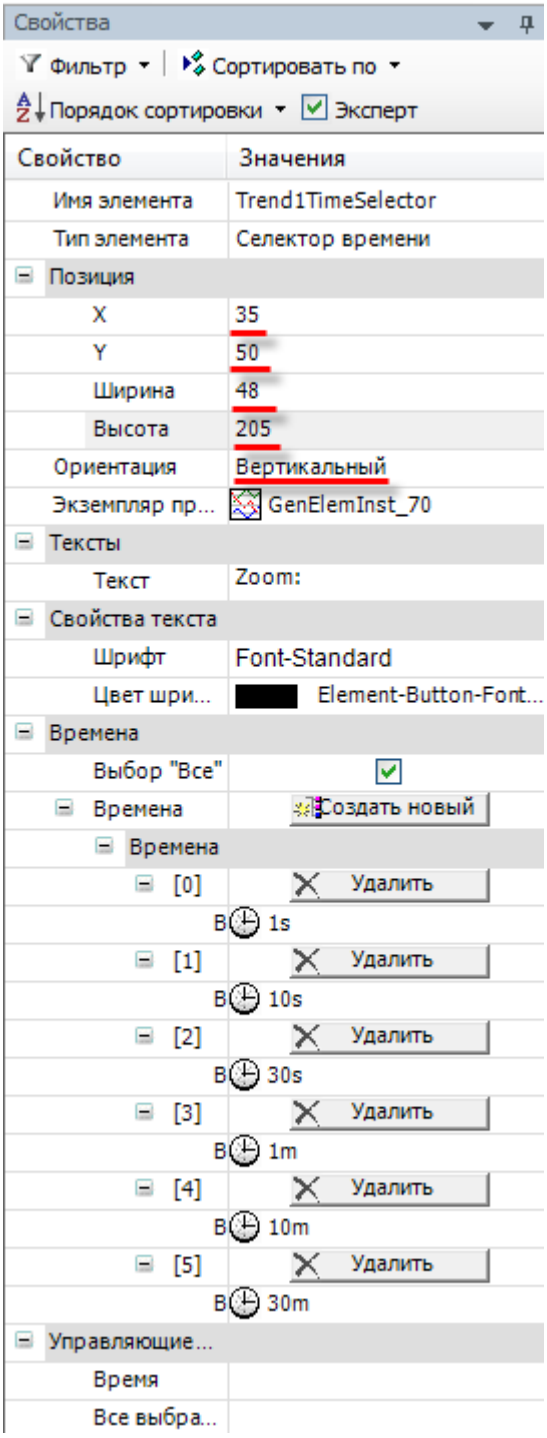


Рис. 7.48. Настройки селектора времени

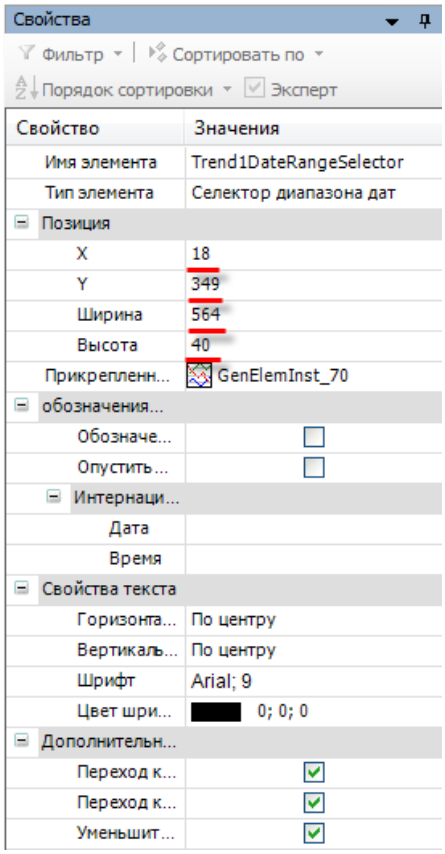


Рис. 7.49. Настройки селектора диапазона дат

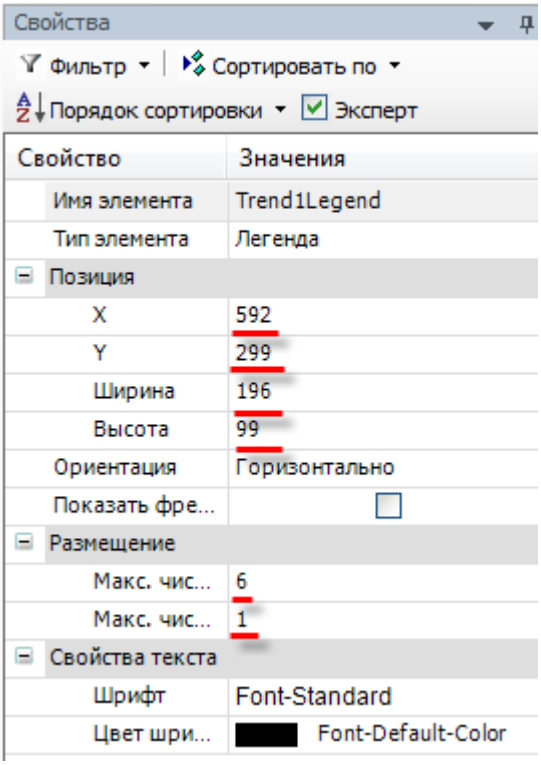


Рис. 7.50. Настройки легенды

Для визуальной настройки тренда (шаг сетки, цвет фона и т.д.) следует нажать на него **ПКМ** и в контекстном меню выбрать пункт **Configure appearance settings of trend**.

Итак, после выполнения действий, описанных в п. 7.3.3, экран **Trend** выглядит следующим образом:

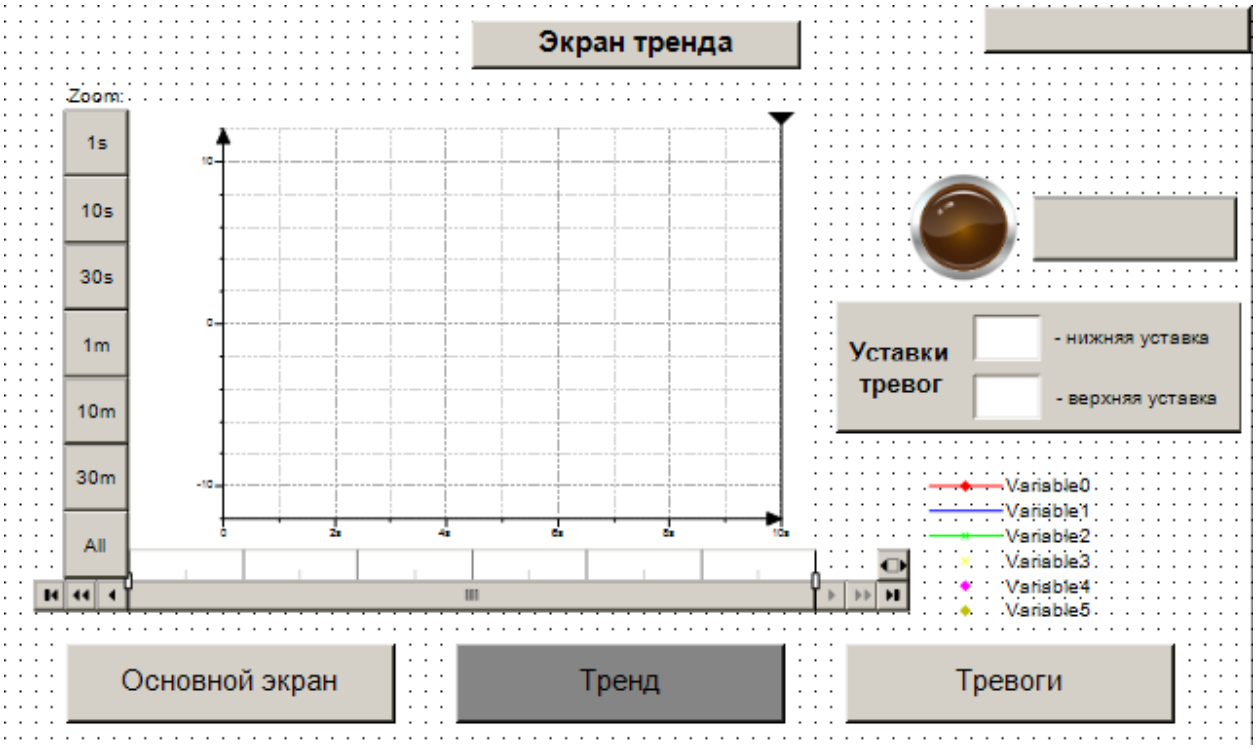


Рис. 7.51. Внешний вид экрана **Trend** (после п. 7.3.3)

7.3.4. Наполнение экрана Alarms_log

Экран **Alarms_log** будет использоваться для отображения **Журнала тревог**, в котором выводятся сообщения о выходе значения температуры за уставки гистерезиса и аварийные уставки.

После выполнения действий, описанных в п. 7.3.2 пп. I (добавление кнопок перехода между экранами и панели системного времени), экран **Alarms_log** выглядит следующим образом:

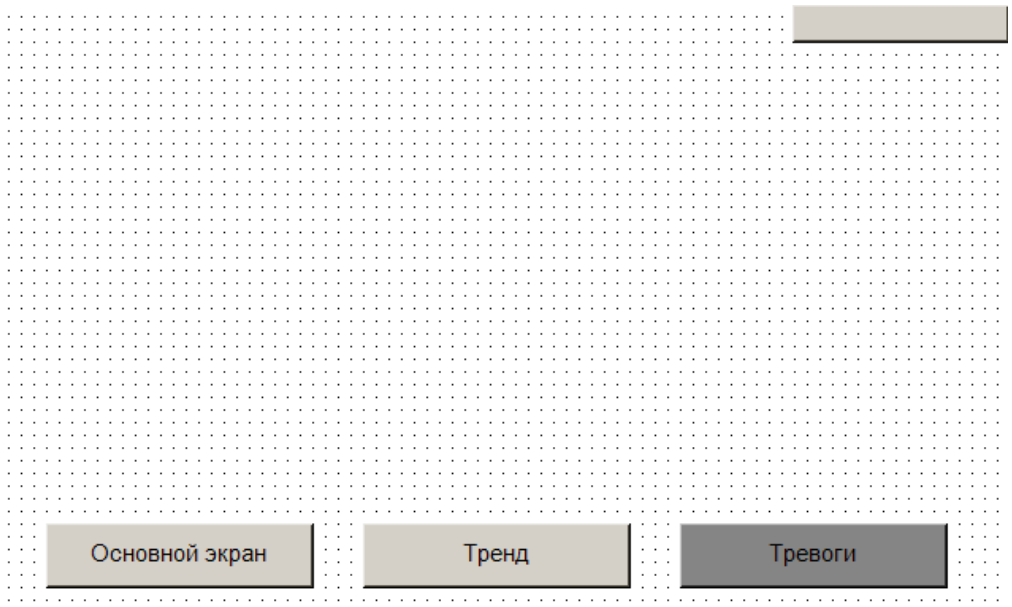


Рис. 7.52. Экран **Alarms_log** (после п. 7.3.2)

Добавим название экрана с помощью элемента **Кнопка**:

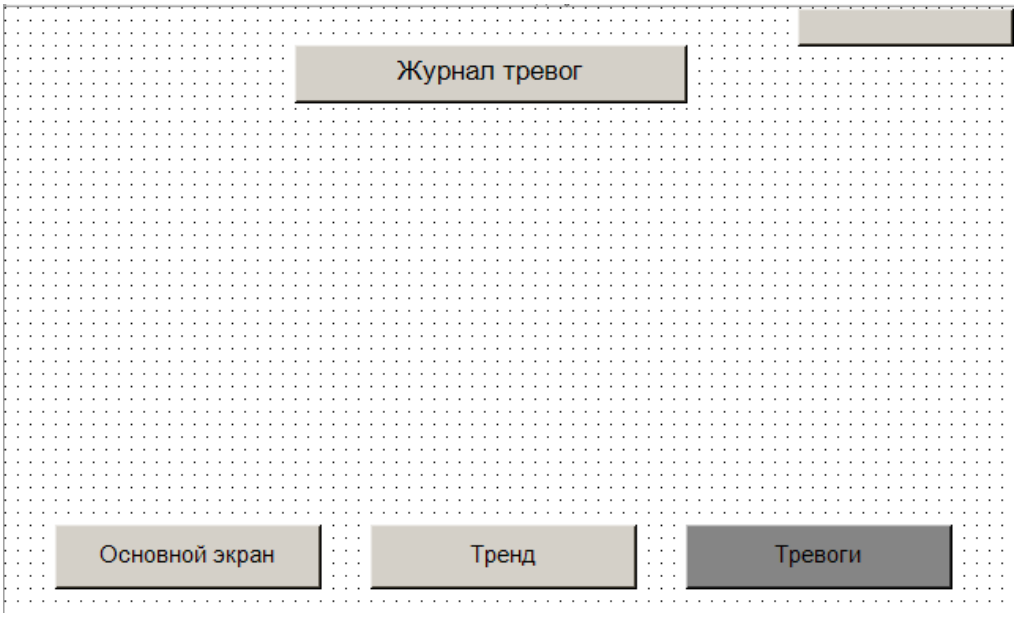


Рис. 7.53. Экран **Alarms_log** – добавление названия экрана

Добавим элемент **Таблица тревог** (вкладка **Менеджер тревог**):

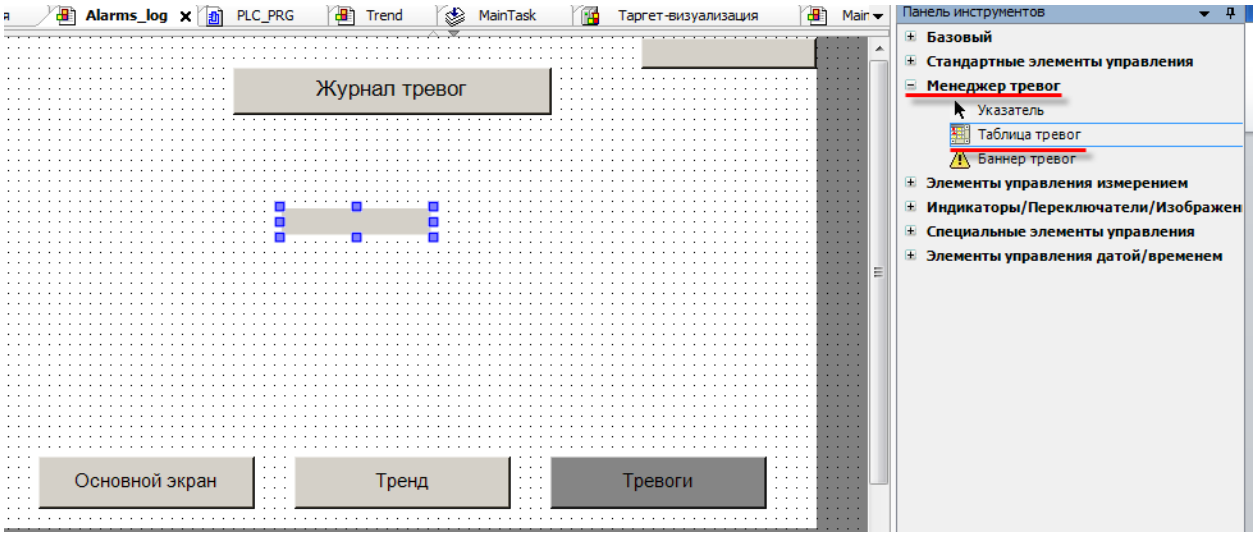


Рис. 7.54. Добавление элемента **Таблица тревог**

Настроим его следующим образом:

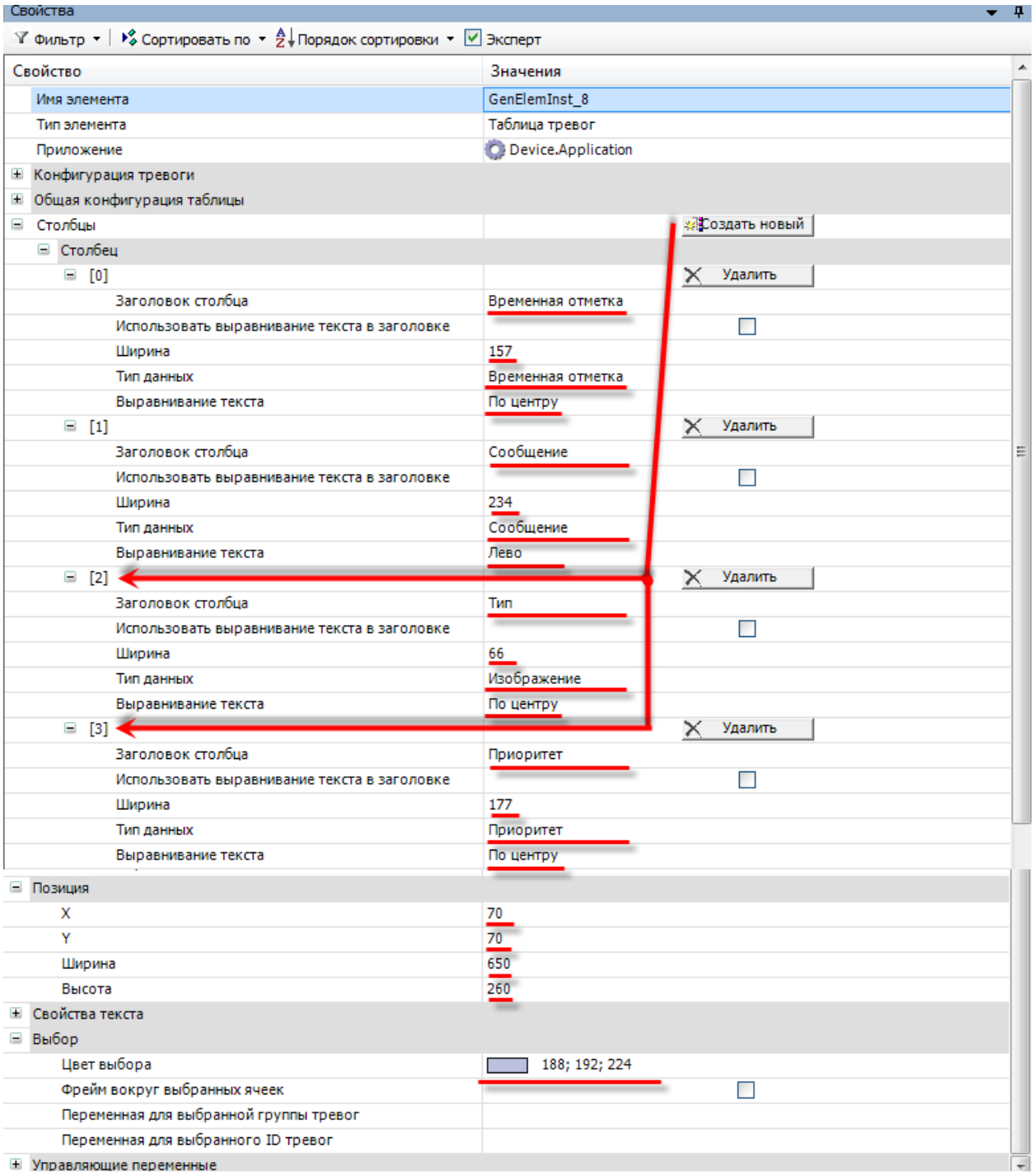


Рис. 7.55. Настройки элемента **Таблица тревог**

В результате **Таблица тревог** будет выглядеть следующим образом:

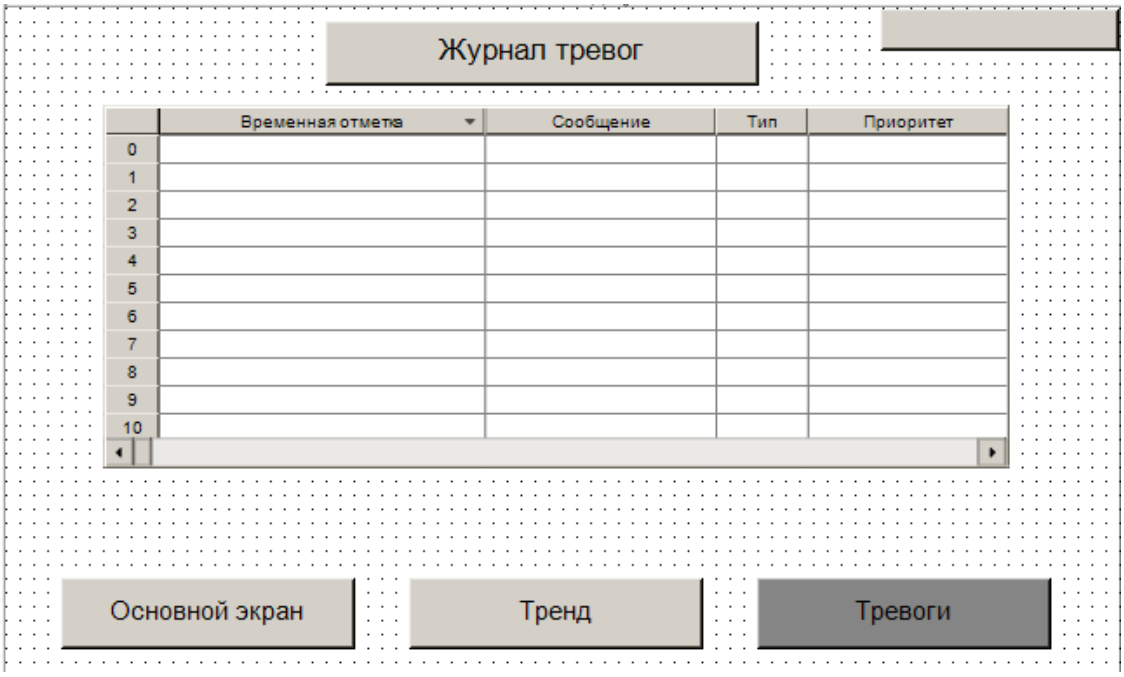


Рис. 7.56. Внешний вид **Таблицы тревог** после настройки

Нажмем на Таблицу тревог **ПКМ** и выберем пункт **Вставить элементы для подтверждения тревог**:

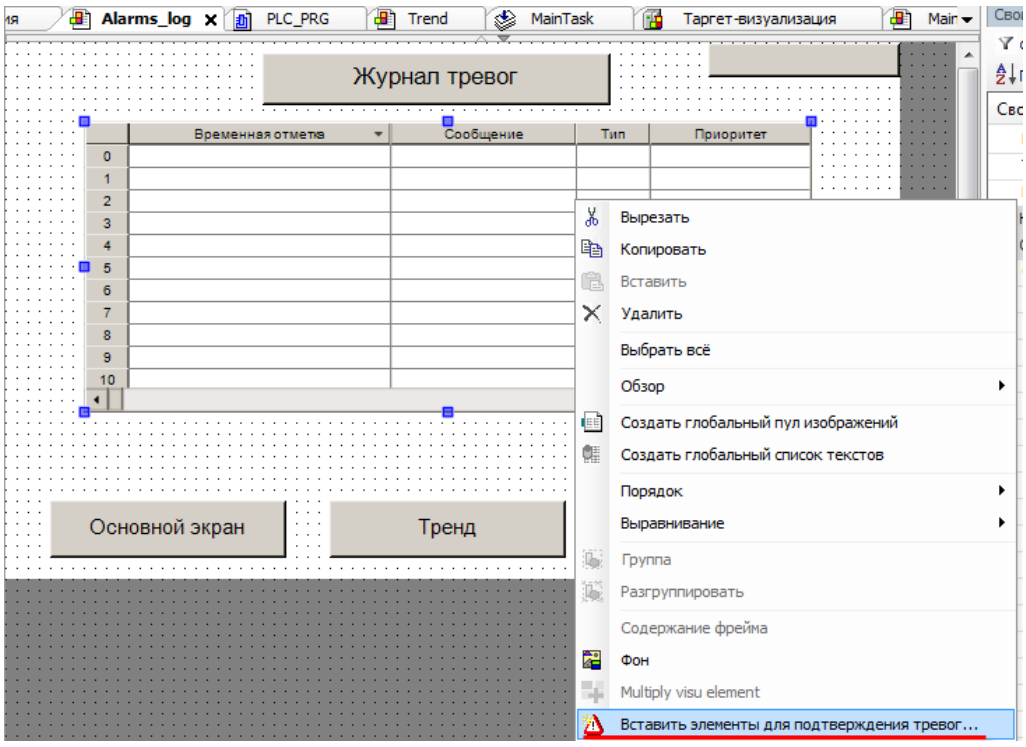


Рис. 7.57. Добавление элементов подтверждения тревог

Появится окно **Мастера таблицы тревог**; оставим его настройки ***по умолчанию***.

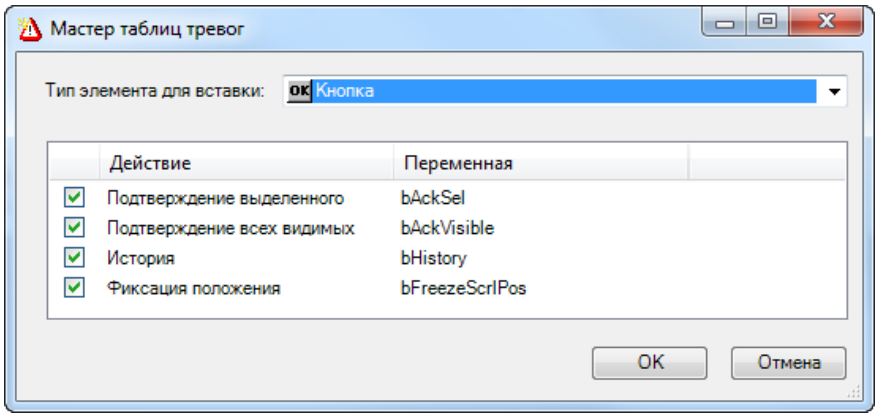


Рис. 7.58. Окно **Мастера таблицы тревог**

В результате появятся элементы подтверждения тревог:

Рис. 7.59. Элементы подтверждения тревог

Изменим их размеры, положение и названия. В результате, экран **Alarms_log** будет выглядеть подобным образом:

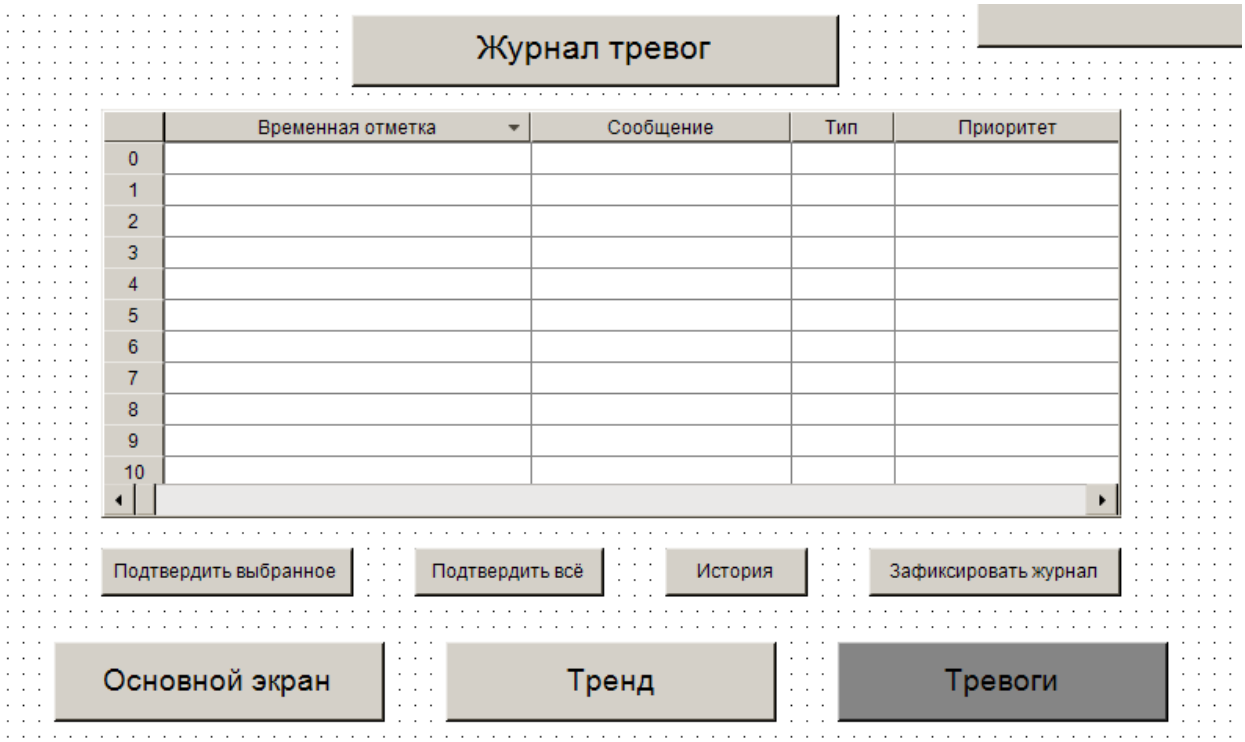


Рис. 7.60. Экран **Alarms_log** (после п. 7.3.4)

Кнопки подтверждения выполняют следующие функции:

1. Кнопка **Подтвердить выбранное** используется для подтверждения (квитирования) выделенной тревоги;

2. Кнопка **Подтвердить всё** используется для подтверждения всех тревог сразу;

3. Кнопка **История** переключает журнал в режим истории для отображения информации о прошедших тревогах;

4. Кнопка **Зафиксировать журнал** запрещает автоматическое перелистывание строк журнала при появлении новых сообщений; это позволяет «заморозить» текущую страницу журнала.

7.3.5. Пул изображений

CODESYS позволяет загружать в проект пользовательские изображения, которые в дальнейшем могут использоваться в процессе разработки экранов визуализации (например, для создания фона экрана). Поддерживается большинство популярных форматов графических файлов, например, .jpg, .png, .bmp, .svg и т.д.).

Загрузка изображений осуществляется через компонент **Пул изображений**. Добавим его в наш проект:

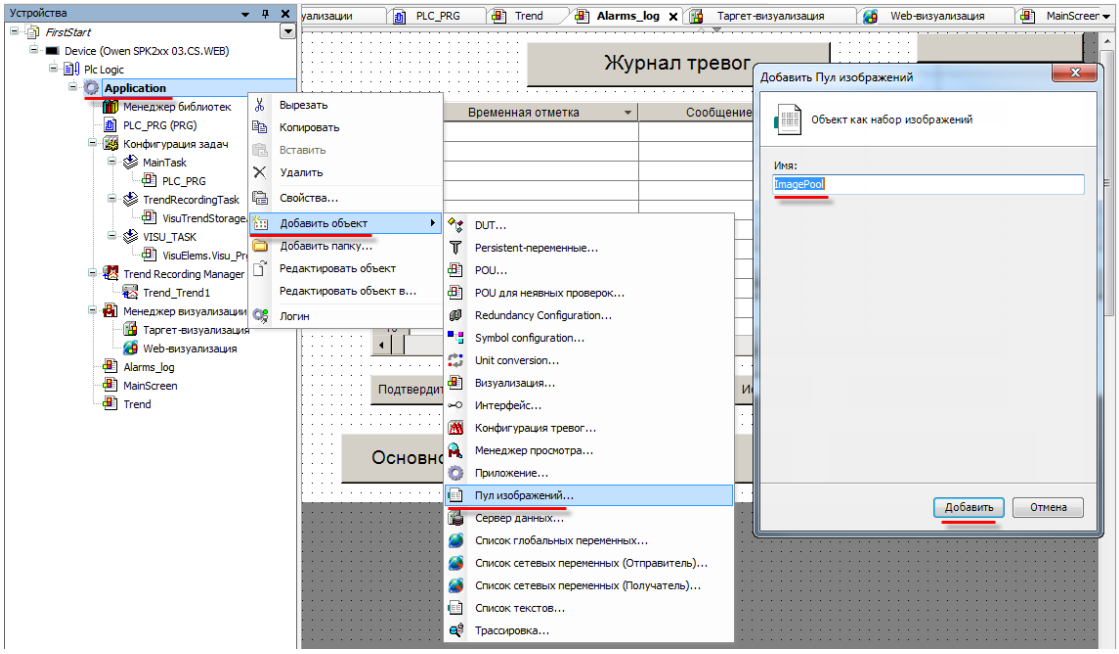


Рис. 7.61. Добавление **Пула изображений**

Откроем компонент двойным нажатием **ЛКМ** на его название:

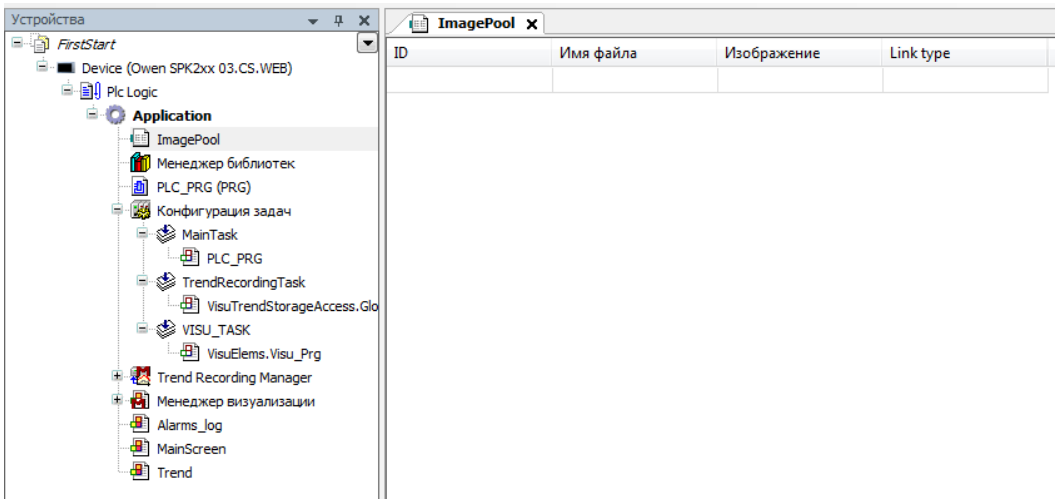


Рис. 7.62. Внешний вид **Пула изображений**

Для добавления изображения следует нажать **ЛКМ** на ячейку **Имя файла** и с помощью появившейся кнопки перейти к выбору файла:

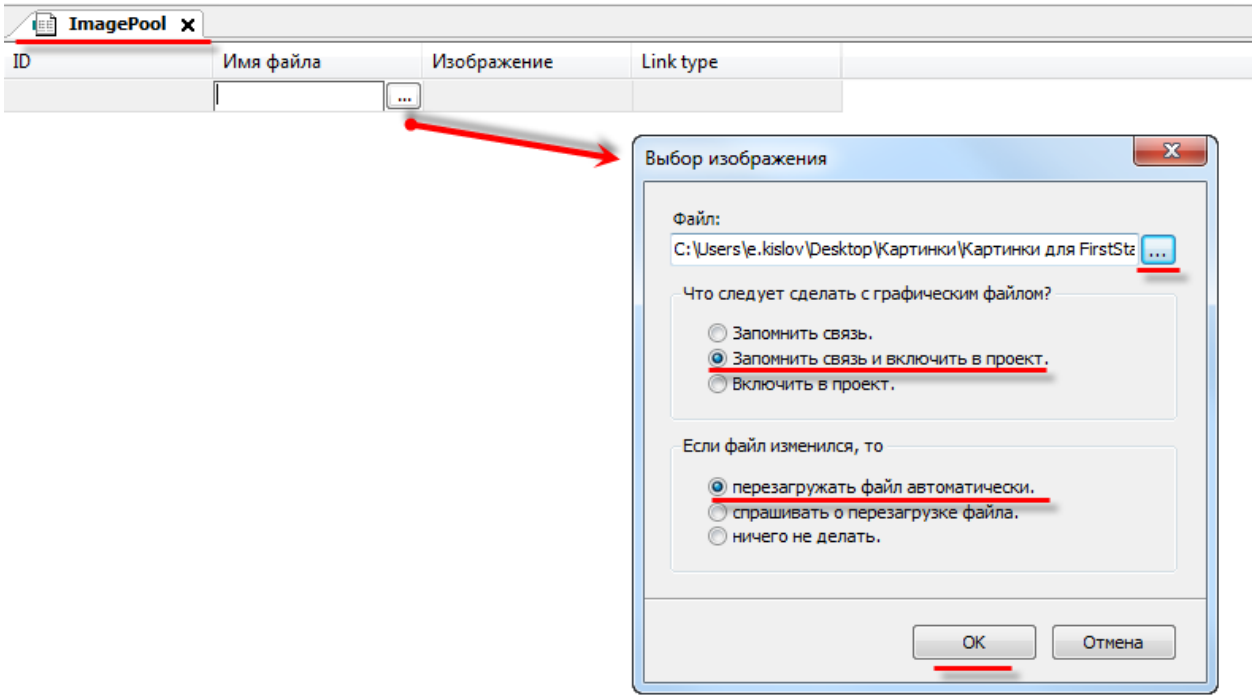


Рис. 7.63. Выбор изображения для загрузки

Укажем путь к графическому файлу, в расположенных ниже меню выберем пункты **Запомнить связь и включить в проект** и **Перезагружать файл автоматически**. Это позволяет не совершать дополнительных операций при изменении изображения – оно будет автоматически меняться в проекте.

После добавления изображения, его пиктограмма отобразится в **Пуле**; также рядом с ним будет указан **идентификатор (ID)** и **тип связи**.

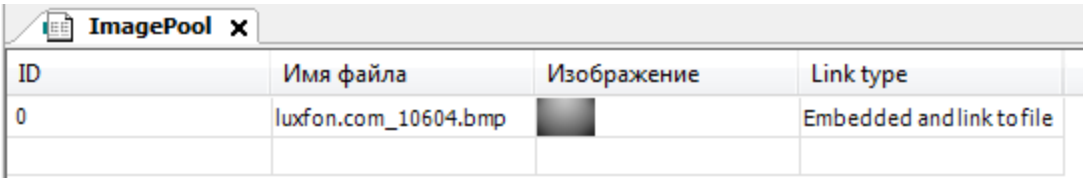


Рис. 7.64. Пул изображений после добавления файла

Теперь, чтобы использовать добавленную картинку в качестве фона экрана визуализации, необходимо нажать **ПКМ** на любое место экрана и в контекстном меню выбрать вкладку **Фон**. В появившемся диалоговом окне следует поставить галочку **Изображение** и с помощью кнопки выбора (перекрывается клавишей Отмена) открыть **Ассистент ввода изображений**:

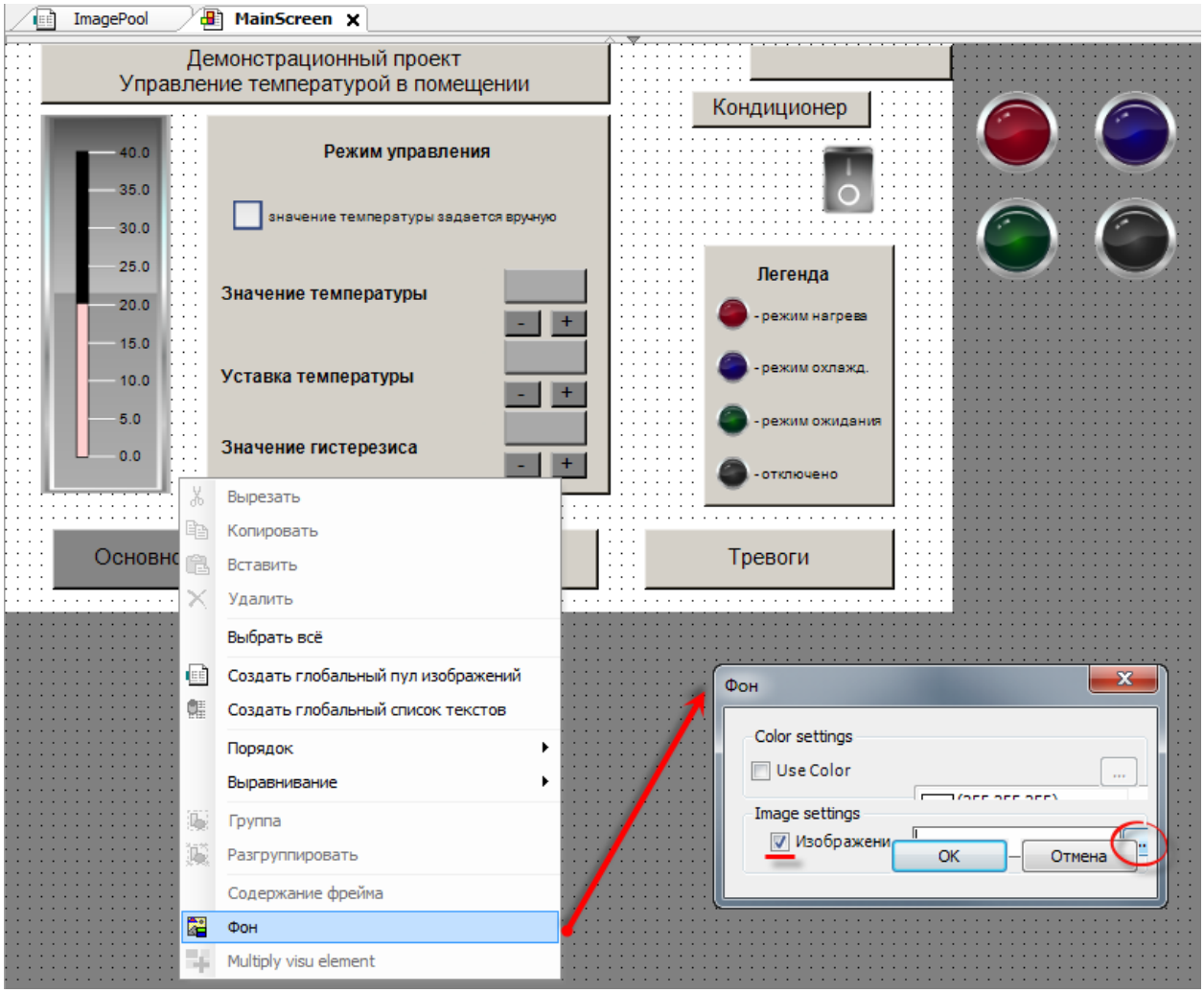


Рис. 7.65. Выбор фона экрана визуализации

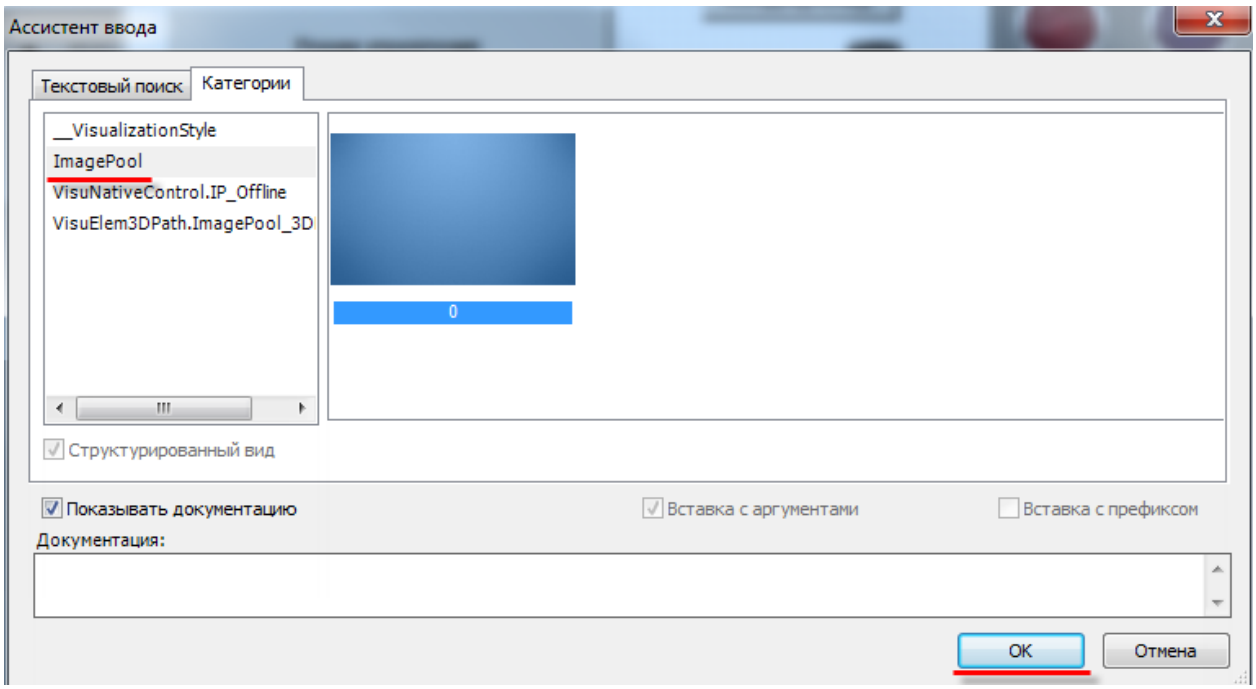


Рис. 7.66. Ассистент ввода изображений

На этом разработка экранов визуализации закончена.

Организацию компонентов в **Панели устройств** можно осуществлять с помощью создания **папок**. Поскольку теперь у нас появилось несколько однотипных компонентов (экраны визуализации), с помощью нажатия **ПКМ** на компонент **Applicaton** в **Панели устройств** создадим новую папку **Визуализация**, и, ***зажав* ЛКМ**, перенесем туда экраны визуализации и компонент **Менеджер визуализации**:

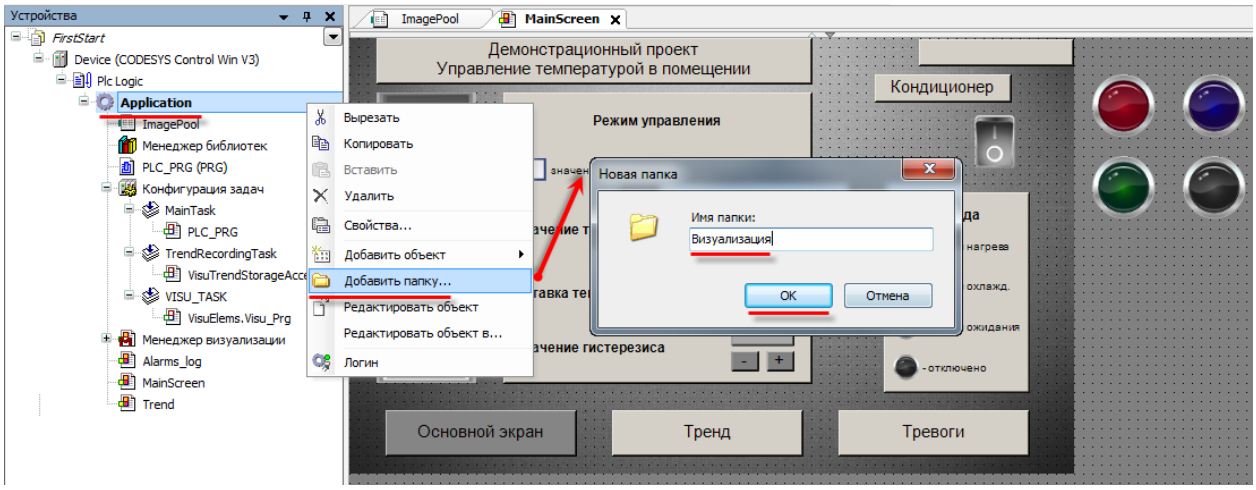


Рис. 7.67. Создание новой папки в приложении **Application** на **Панели устройств**

Теперь **Панель устройств** нашего проекта будет выглядеть так:

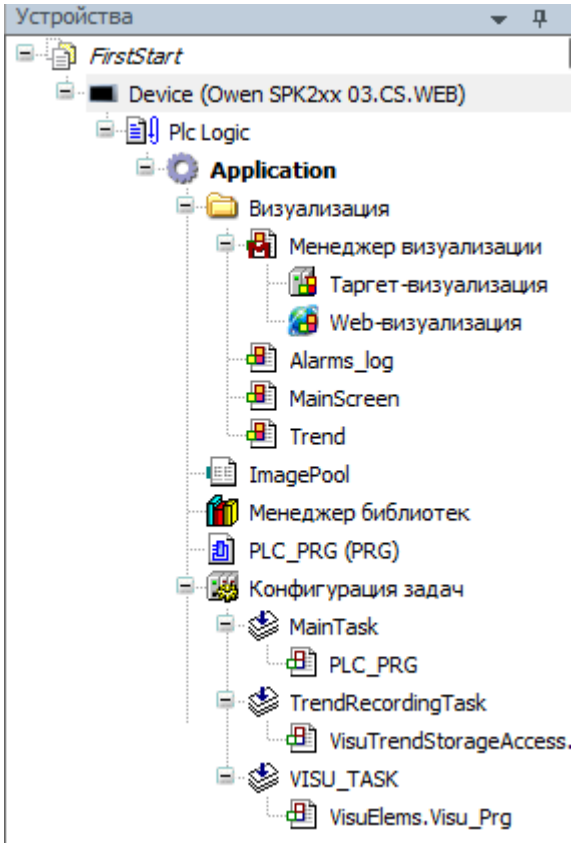


Рис. 7.68. Панель устройств (после п. 7.3)

Экраны визуализации, созданные по указаниям пункта 7.3, выглядят следующим образом:

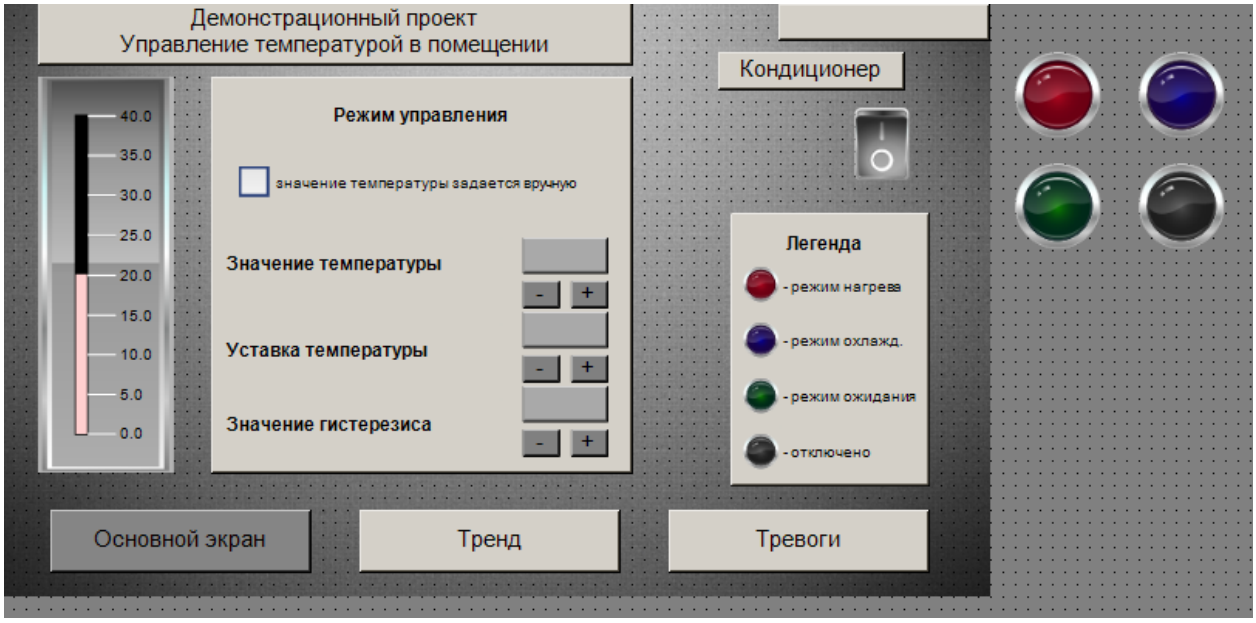


Рис. 7.69. Экран **MainScreen** (после п. 7.3)

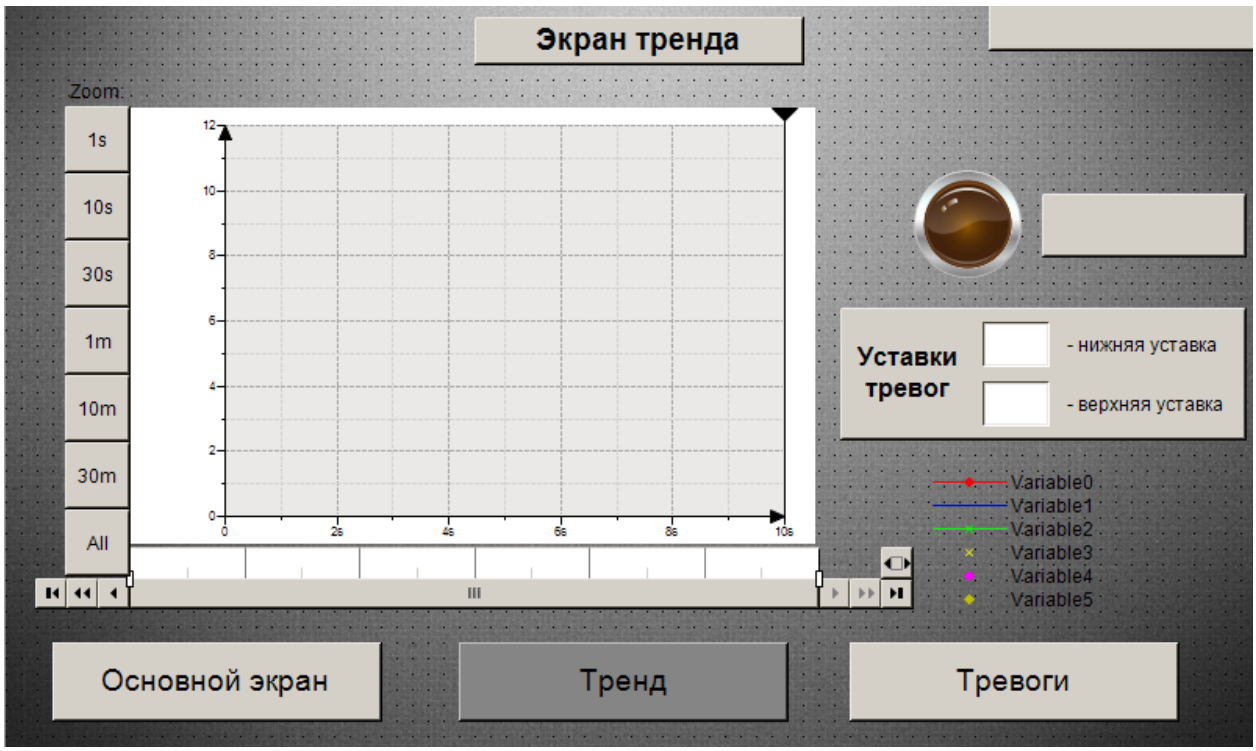


Рис. 7.70. **Экран Trend** (после п. 7.3)

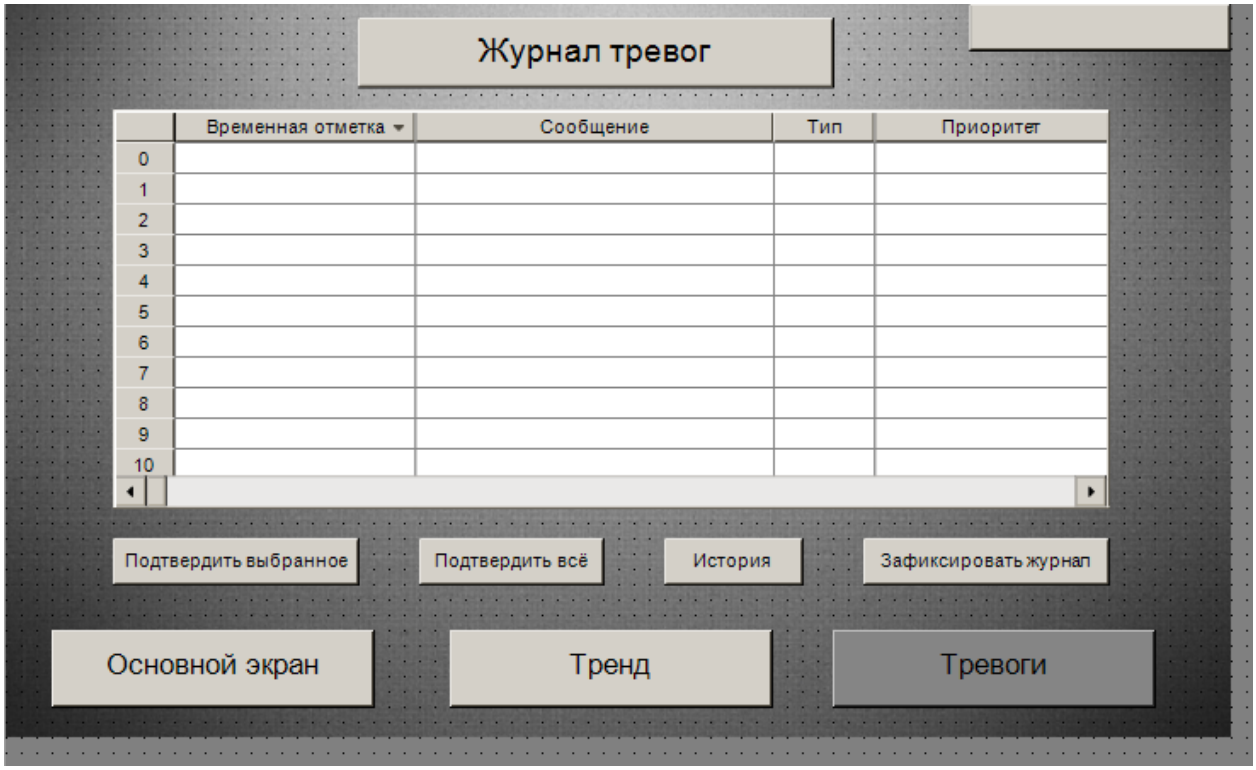


Рис. 7.71. Экран **Alarms_log** (после п. 7.3)

Напоминаем, что функциональная настройка элементов и привязывание к ним переменных будут описаны в п. 7.5.

7.4. Разработка программ

7.4.1. POU и их типы. Языки программирования МЭК 61131-3. Структура приложения проекта FirstStart

Пользовательский проект может содержать одно или несколько приложений, которые состоят из компонентов, называемых **POU** (Program Organisation Unit). POU, которые используются для разработки **программной части проекта**, принадлежат к определенному **типу** и характеризуются **языком программирования**.

Существует **три типа** «программных» POU:

1. **Программа** - это POU, при выполнении которого возвращается одно или несколько значений. После выполнения программы все значения сохраняются до ее следующего запуска. Программа может быть вызвана другим POU, но вызов программы в функции (см. ниже) не допускается;

2. **Функция** - это POU, при выполнении которого возвращается только **одно** значение. Функция может быть вызвана другим POU;

3. **Функциональный блок** - это POU, который при выполнении возвращает одно или несколько значений. В отличие от функции, функциональный блок имеет память, т.е. значения его аргументов могут сохраняться. Вызов функционального блока осуществляется через его **экземпляр**, т.е. копию.

В данной главе мы рассмотрим использование всех трех типов программных компонентов. ***Напомним***, что рассматриваемая нами задача является учебным примером, поэтому некоторые решения не являются целесообразными в реальной жизни.

POU может быть написан на одном их шести **языков программирования** - пять из них входят в состав стандарта **МЭК 61131-3**, шестой является расширением языка FBD:

1. **IL (Instruction List**) - текстовый язык аппаратно-независимый низкоуровневый ассемблероподобный язык, в последнее время практически вышедший из употребления;

2. **LD (Ladder Diagram)** - графический язык релейно-контактных схем, подходящий для программирования релейной логики;

3. **SFC (Sequential Function Chart)** - графический высокоуровневый язык, созданный на базе математического аппарата сетей Петри. Описывает последовательность состояний и условий переходов;

4. **ST (Structured Text)** - текстовый паскалебодобный язык программирования;

5. **FBD (Functional Block Diagram)** - графический язык, программа на котором

состоит из последовательно соединенных **функциональных блоков**;

6. **CFC (Continuous Function Chart)** - расширение языка FBD; позволяет произвольно задавать порядок выполнения блоков и создавать обратную связь между ними.

Выбор языка программирования осуществляется при добавлении в проект соответствующего программного компонента.

Написание программ на любом из языков происходит в соответствующих редакторах, каждый из которых имеет две области - область определения переменных (верхняя) и область программного кода (нижняя).

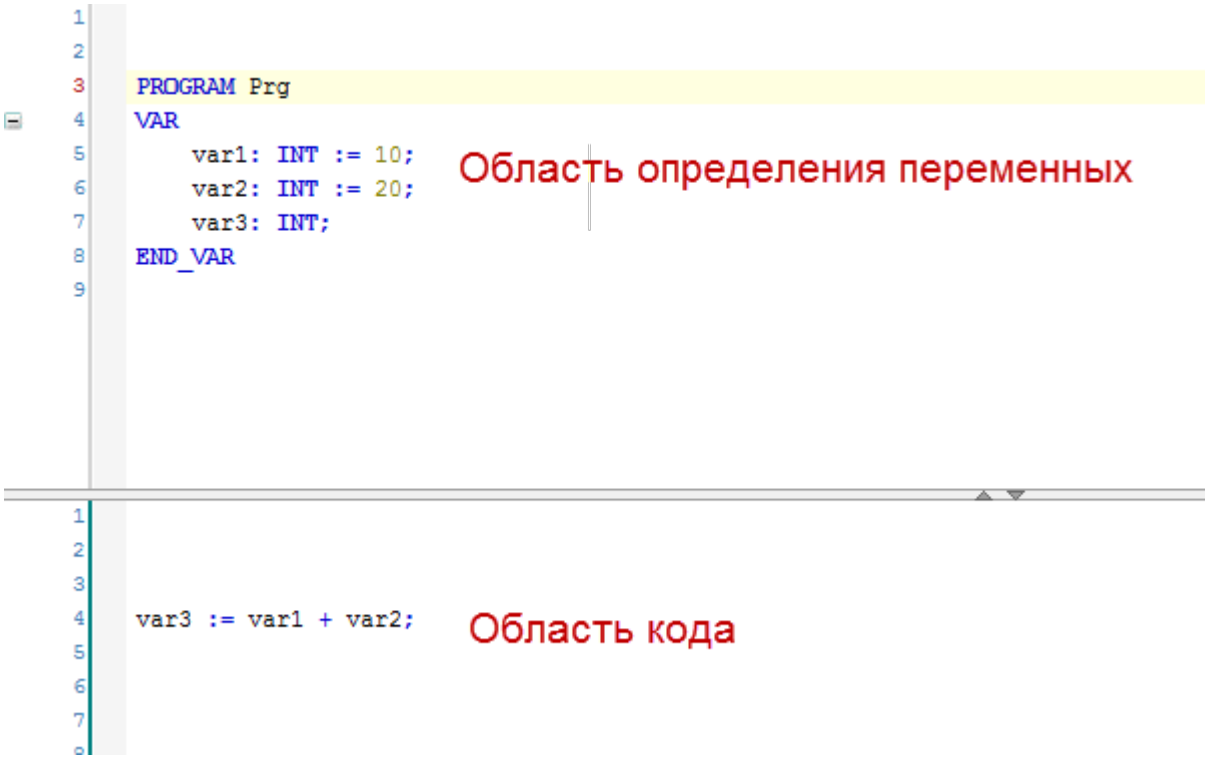


Рис. 7.72. Области редактора программирования

В рассматриваемом примере мы будем использовать языки **ST** и **CFC** как наиболее распространенные и востребованные.

Итак, зададимся следующей структурой программной части приложения:

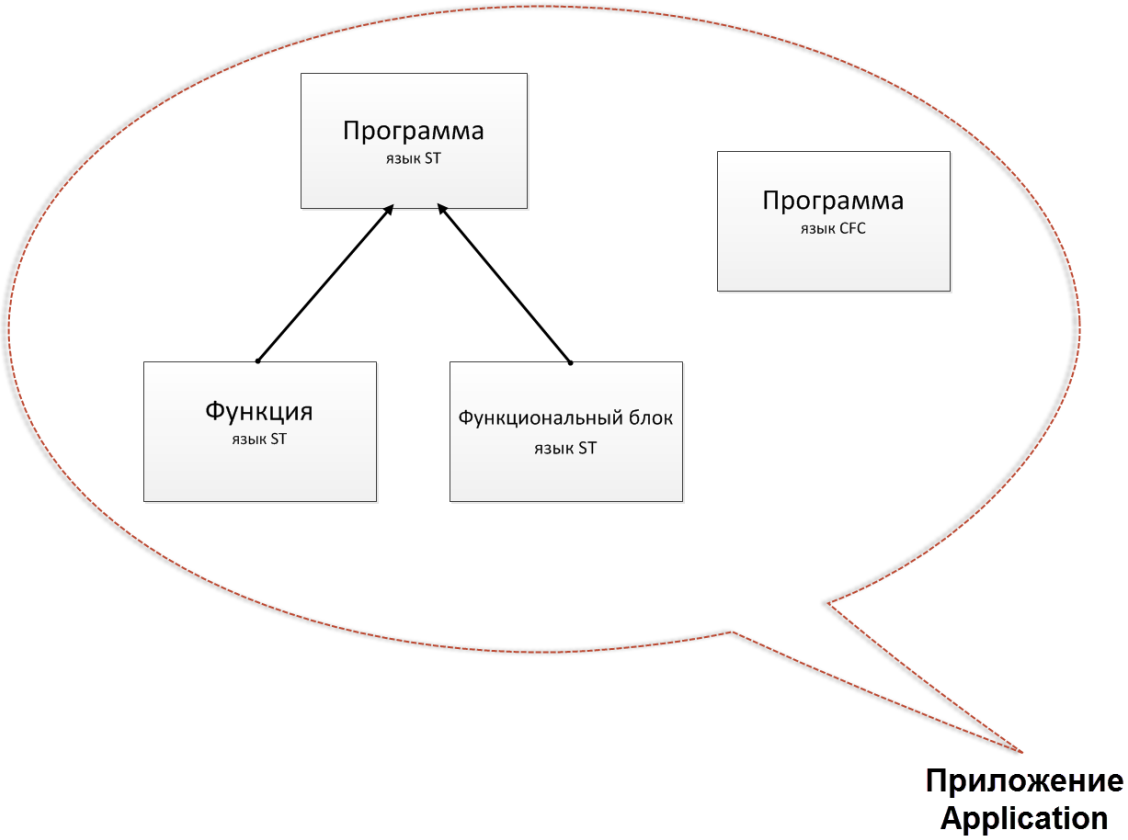


Рис. 7.73. Структура программ приложения Application

7.4.2. Виды переменных. Типы данных. Определение глобальных переменных

Переменные необходимы для ввода, хранения и вывода данных программ. Существует два типа переменных - **глобальные** (область видимости - весь проект) и **локальные** (область видимости - конкретный **POU**). Имена глобальных и локальных переменных ***могут совпадать***, при этом локальная переменная ***имеет приоритет*** перед глобальной в **своем POU** (следует соблюдать осторожность при использовании одинаковых имен переменных).

Переменная может обладать одним или несколькими атрибутами, из которых необходимо отметить **retain** - значения таких переменных после выключения контроллера сохраняются в **энергонезависимой памяти** (объем энергонезависимой памяти для всех моделей СПК - 4 кб). ***Обратите внимание***, что переменные типа **persistent** ***не поддерживаются*** контроллерами СПК.

Переменная обязательно принадлежит к какому-либо **типу данных**. Существует три вида типов данных - стандартные (определенные МЭК 61131-3), пользовательские, а также экземпляры функциональных блоков. Характеристики стандартных типов данных приведены ниже:

Табл. 1. Характеристики стандартных типов данных

Тип данных	Нижний предел	Верхний предел	Размер памяти
Логические типы данных			
BOOL	FALSE	TRUE	8 бит
Целочисленные типы данных			
BYTE	0	255	8 бит
WORD	0	65535	16 бит
DWORD	0	4294967295	32 бита
LWORD	0	2^{64}-1	64 бита
INT	-32768	32767	16 бит
UINT	0	65535	16 бит
SINT	-128	127	8 бит
USINT	0	255	8 бит
DINT	-2147483648	2147483647	32 бит
UDINT	0	4294967295	32 бит
LINT	-2^{63}	2^{63}-1	64 бит
ULINT	0	2^{64}-1	64 бит
Типы данных с плавающей точкой			
REAL	$1.175494351e^{-38}$	$3.402823466e^{38}$	32 бит
LREAL	$2.225073858507201e^{-308}$	$1.7976931348623158e^{308}$	64 бит
Строковые типы данных			
STRING WSTRING	отсутствие символов	нет (неограниченное количество символов); строковые функции могут работать со строками, чья длина не превышает 255 символов	не ограничен
Временные типы данных (подробнее о формате и ограничениях см. в справке CODESYS)			
TIME	0h0m0s0ms	49d17h2m37s295ms	32 бита
TIME_OF_DAY	0:0:0.0	23:59:59.999	32 бита
DATE	1970-1-1	2106-2-7	32 бита
DATE_AND_TIME	1970-1-1-0:0:0.0	2106-2-7-6:28:15	32 бита

Для определения **глобальных переменных** создадим соответствующий компонент – **Список глобальных переменных** с названием **GlobalVars**:

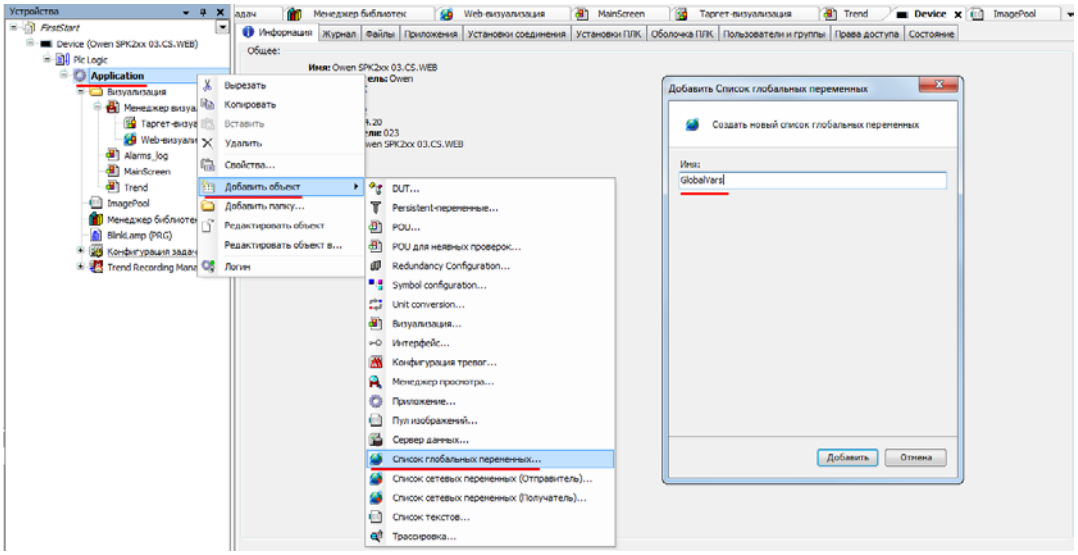


Рис. 7.74. Создание списка глобальных переменных

Определение глобальных переменных осуществляется между ключевыми словами **VAR_GLOBAL** и **END_VAR**, глобальных **retain** (энергонезависимых) переменных - между **VAR_GLOBAL RETAIN** и **END_VAR** соответственно (см. рис. 7.75).

Синтаксис определения переменной следующий (угловые скобки использованы для наглядности, при работе со средой программирования использовать их не следует):

<Имя переменной>: <Тип> {:=<начальное значение>};

где

<Имя переменной> – индивидуальное название переменной, на которое накладываются определенные ограничения, приведенные в справке CODESYS. Пока же достаточно пояснить, что регистр имени переменной не учитывается, оно не должно содержать пробелов, специальных и кириллических символов, не должно содержать более одного подчеркивания подряд, не должно совпадать с ключевыми словами (например, TIME), локально не могут быть объявлены переменные с совпадающими именами;

<Тип> – тип переменной. Характеристики стандартных типов приведены в табл. 1;

<начальное значение> – начальное значение переменной (по умолчанию равно нулю);

“**:=**” – оператор присваивания;

“**;**” – символ, которым завершается любая процедура определения, присваивания, вызова функции и т.п.

Синтаксис определения **строковых переменных** несколько отличается от определения переменных других типов: после названия типа в скобках указывается **максимальная длина строки**, которая определяет количество резервируемой памяти (без указания – 80 символов). Содержимое строки для типа **STRING** (используемого для латиницы) указывается в **одиночных кавычках** (‘<содержимое строки>’), а типа **WSTRING** (кириллица) – в **двойных** (“<содержимое строки>”).

Список глобальных переменных, которые мы будем использовать в наших программах, приведен ниже:

Device | GlobalVars | Менеджер библиотек | Конфигурация задач

```
VAR_GLOBAL
    temp_real:          REAL; (*температура помещения, полученная с датчика*)
    temp_user:          REAL; (*температура помещения, введенная пользователем*)

    temp:               REAL; (*текущая температура в помещении для использования внутри цикла*)

    temp_hyst_high:     REAL; (*температура верхней границы гистерезиса*)
    temp_hyst_low:      REAL; (*температура нижней границы гистерезиса*)

    measure_mode:       BOOL :=TRUE; (*режим измерения:          0 - с датчика,  1 - ввод пользователем*)
    control_mode:       BOOL;        (*режим управления:         0 - охлаждение, 1 - нагрев*)
    regulator_state:    BOOL;        (*состояние регулятора:     0 - выключен,   1 - включен*)
    conditioner_power: BOOL;         (*состояние кондиционера:   0 - выключен,   1 - включен*)


    yellow_lamp_name:  WSTRING(20) :="Введите название"; (*название сигнальной лампы с экрана Тренда*)
END_VAR


VAR_GLOBAL RETAIN
    temp_ust:           REAL :=20; //уставка температуры

    hyst:               REAL :=5;  //значение гистерезиса в процентах

    alarm_high:         REAL;      //верхняя граница сигнала тревоги
    alarm_low:          REAL;      //нижняя граница сигнала тревоги
END_VAR
```

Рис. 7.75. Определение глобальных переменных

Локальные переменные будут описаны в процессе разработки программ.

Как можно заметить, для написания **комментариев** используются конструкции типа (*<комментарий>*) и //<комментарий>. Комментарий с конструкцией первого типа может занимать несколько строк, второго типа - только одну строку.

7.4.3. Программа на языке CFC (мигание лампы). Подключение дополнительных библиотек

Создание **мигающего индикатора** (в общем случае - логической переменной, состояния которой меняются с определенной частотой) - довольно распространенная задача при разработке автоматических систем. В нашем примере такая программа будет использоваться для создания аварийной лампы, мигающей при выходе значения температуры за **аварийные уставки**.

В нашем проекте уже имеется пустой **POU** типа **программа** на языке **CFC** с названием **PLC_PRG**. С помощью функции **Refactoring** изменим его название на **BlinkLamp**:

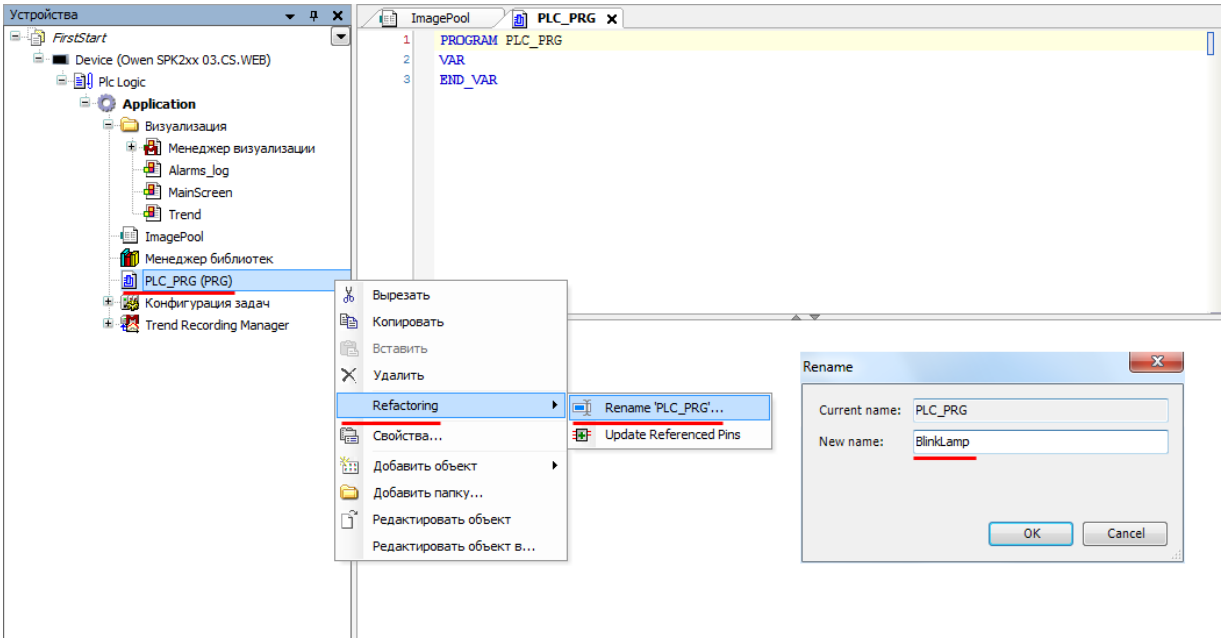


Рис. 7.76. Изменение название программы (refactoring)

Появится окно, в котором указаны компоненты, на которые повлияет смена названия:

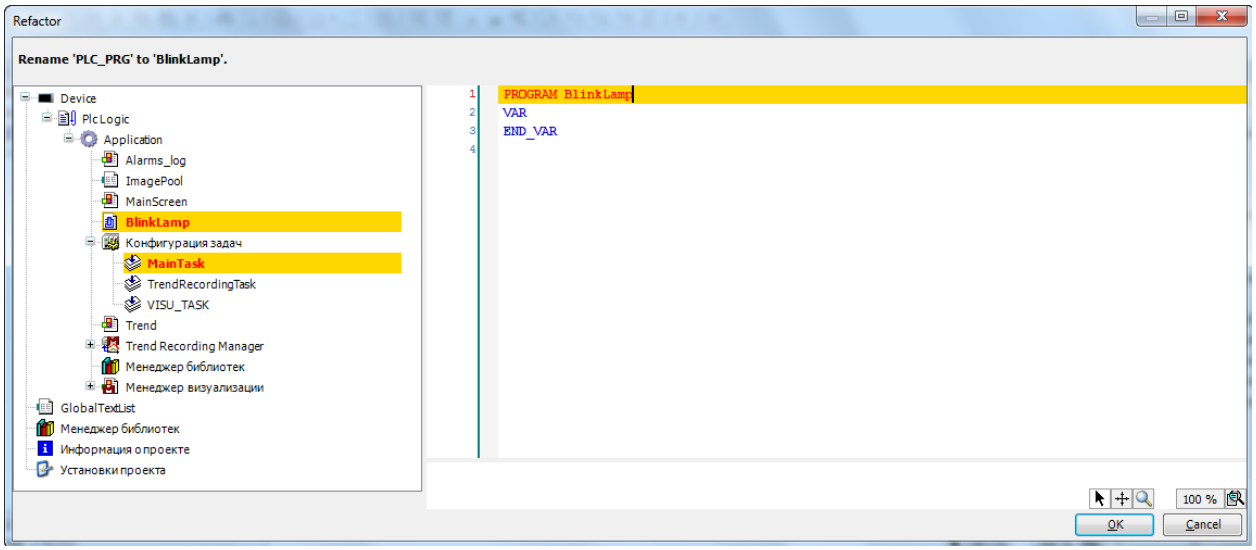


Рис. 7.77. Диалоговое окно refactoring’а

При разработке программы нам понадобится **функциональный блок Blink** из **библиотеки Util**, которая по умолчанию **не включена** в проект. Чтобы добавить ее, выделим компонент **Менеджер библиотек** и нажмем кнопку **Добавить библиотеку** (предварительно библиотека должна быть установлена, см. п.2):

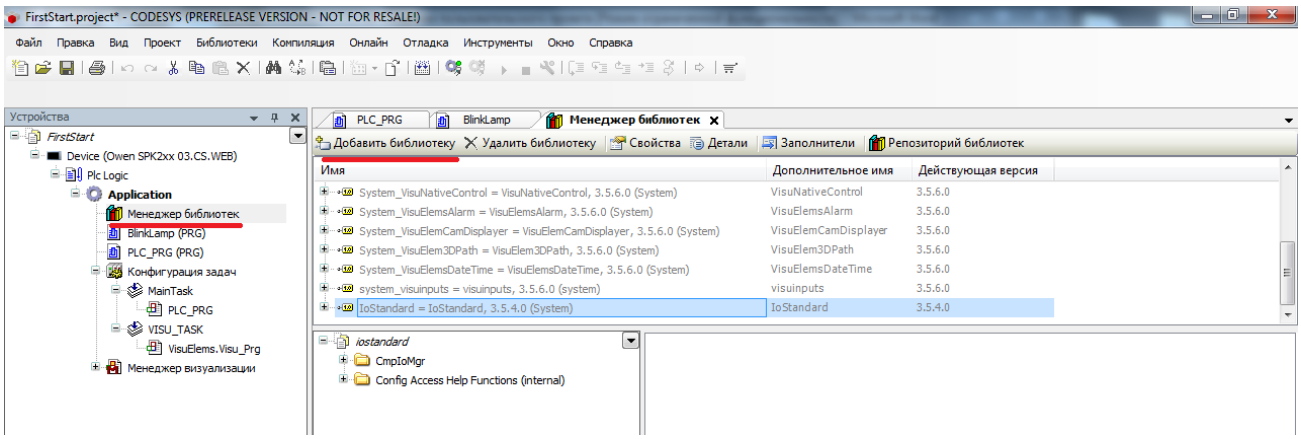


Рис. 7.78. Добавление библиотеки

Выберем нужную библиотеку из списка и нажмем **ОК**.

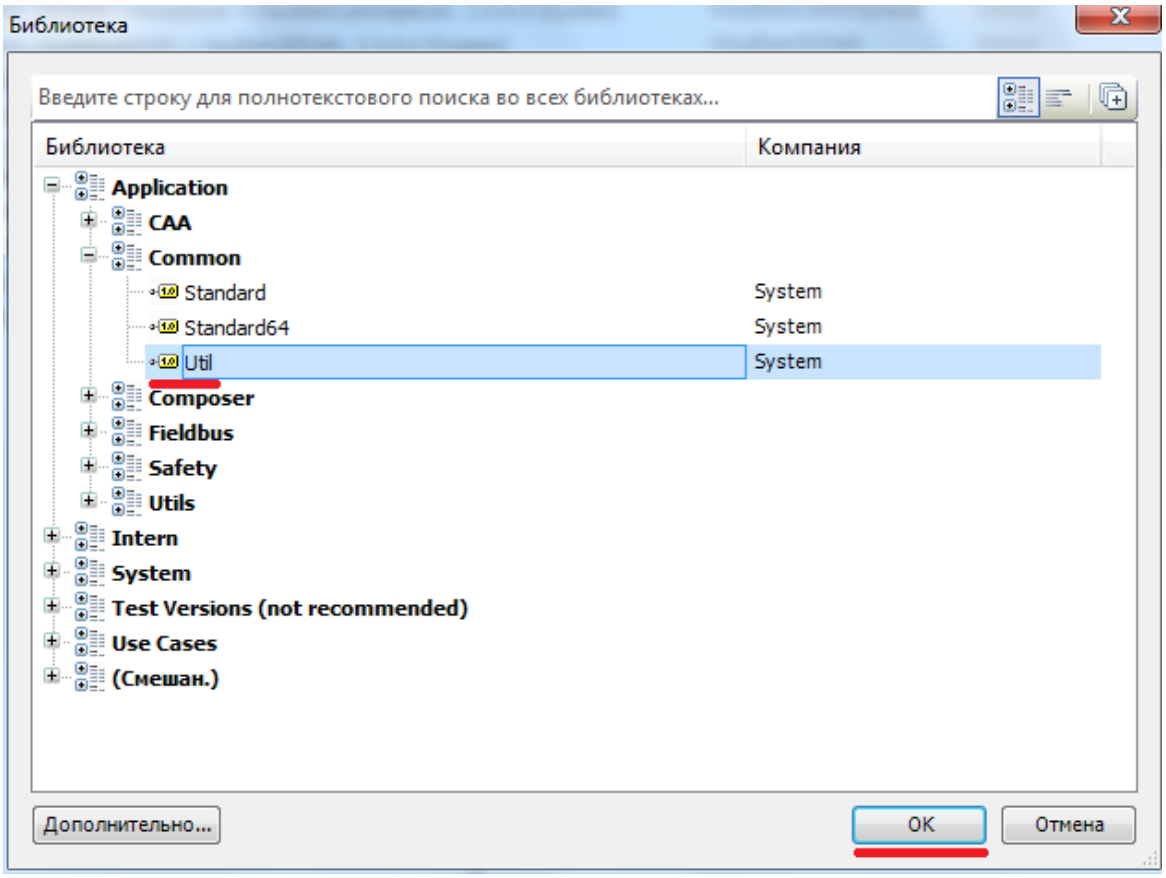


Рис. 7.79. Выбор библиотеки

Библиотека **Util** появится в списке используемых библиотек:

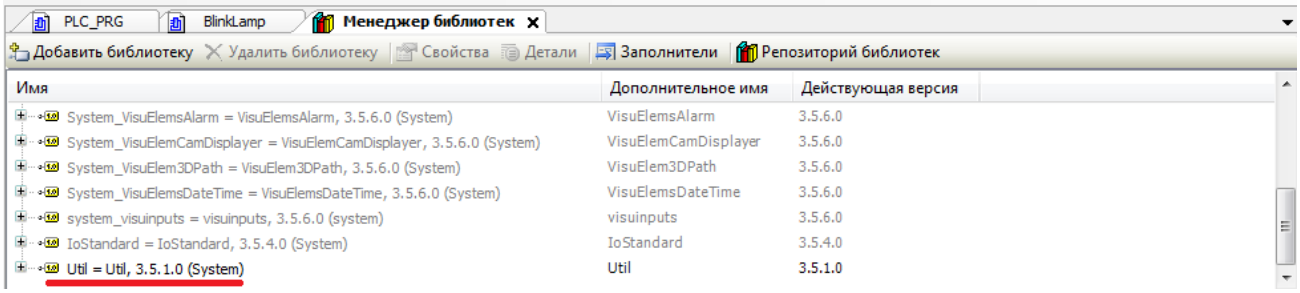


Рис. 7.80. Библиотека **Util**

Откроем программу **BlinkLamp**, однократным нажатием **ЛКМ** выделим на **Панели инструментов** редактора элемент **«Элемент»** и однократным нажатием **ЛКМ** разместим его **в рабочей области**.

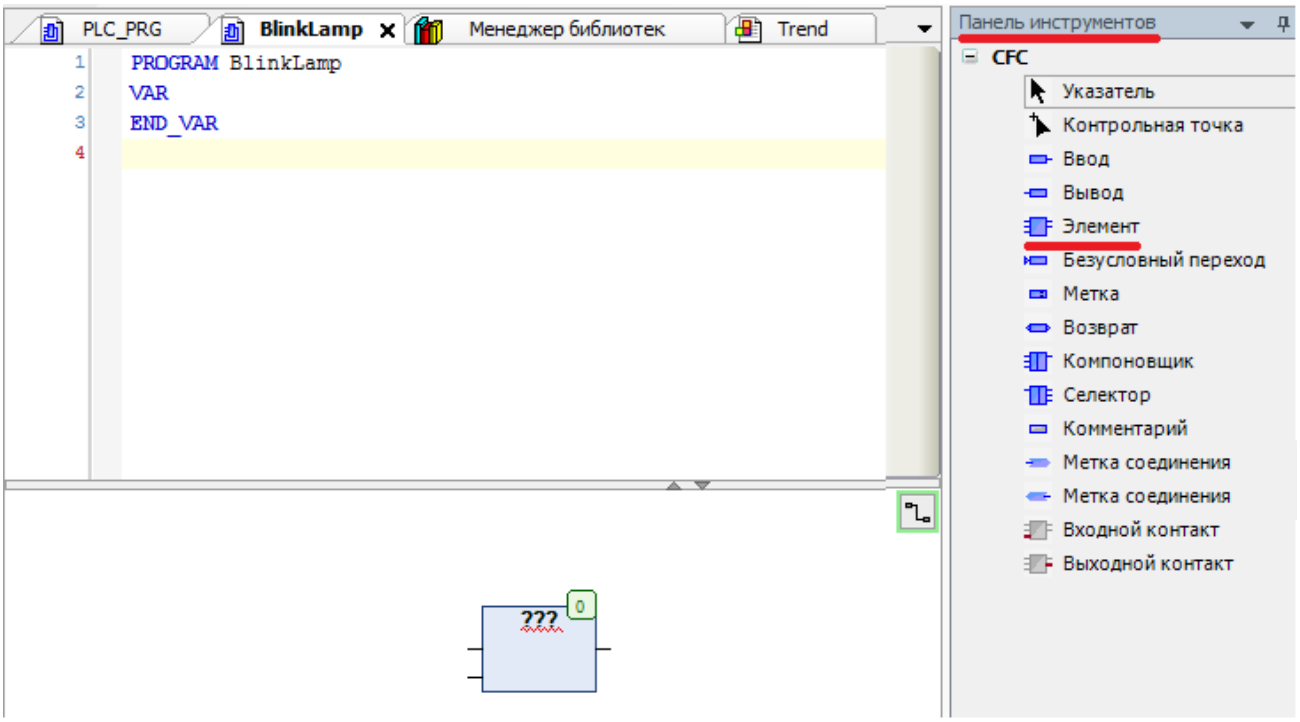


Рис. 7.81. Добавление элемента в редакторе CFC

Введем вместо «**???**» тип функционального блока – **BLINK**. Также можно выбрать функциональный блок из списка доступных с помощью нажатия на контекстную кнопку «**…**», которая откроет **Ассистент ввода**.

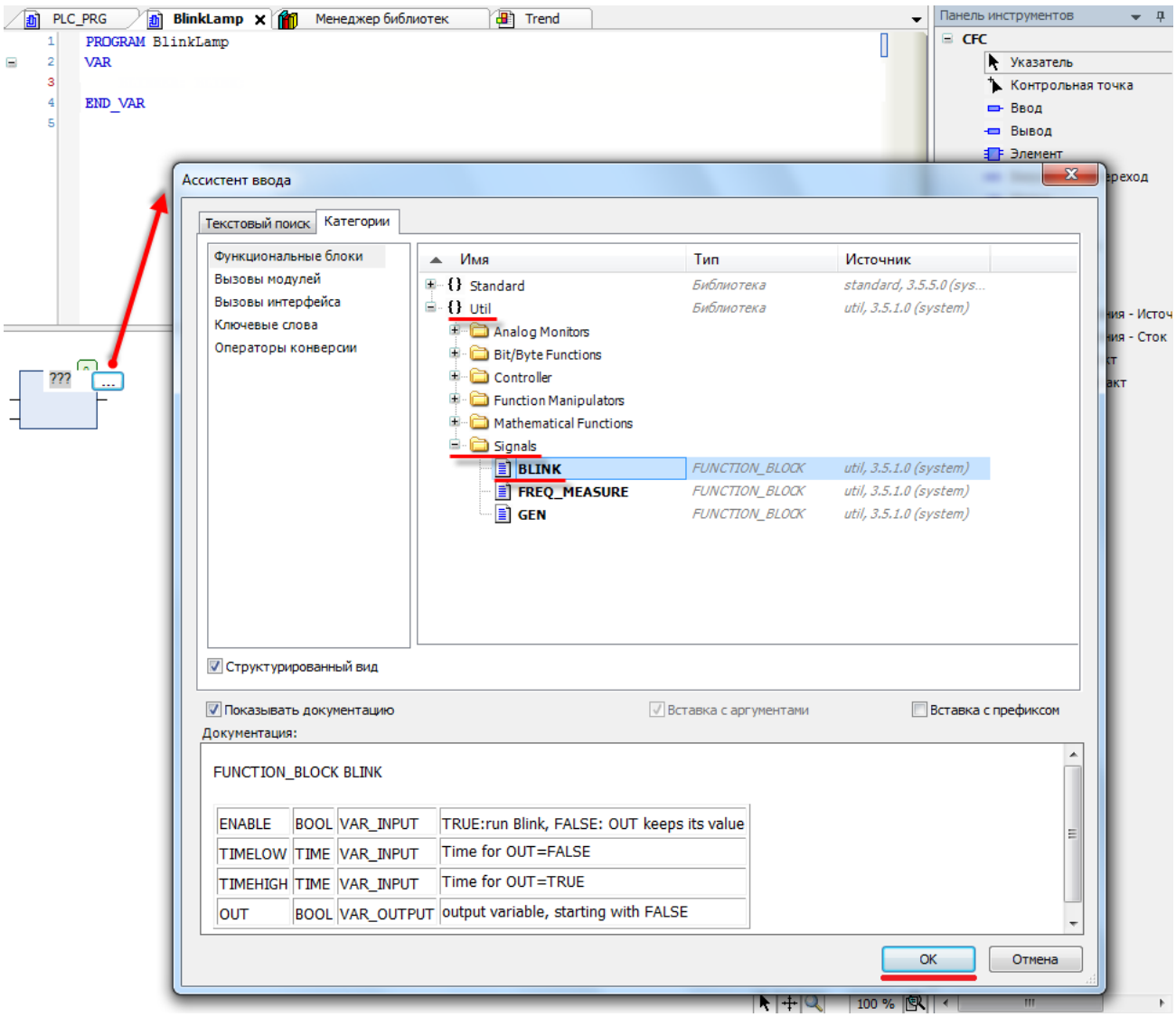


Рис. 7.82. Выбор функционального блока

В результате мы получим **экземпляр Blink_0** функционального блока **BLINK**. Поменяем название экземпляра функционального блока на **BLINKER** и нажмем **ОК**, что приведет к появлению **Ассистента автообъявления** (т.к. мы еще не определили экземпляр функционального блока как переменную).

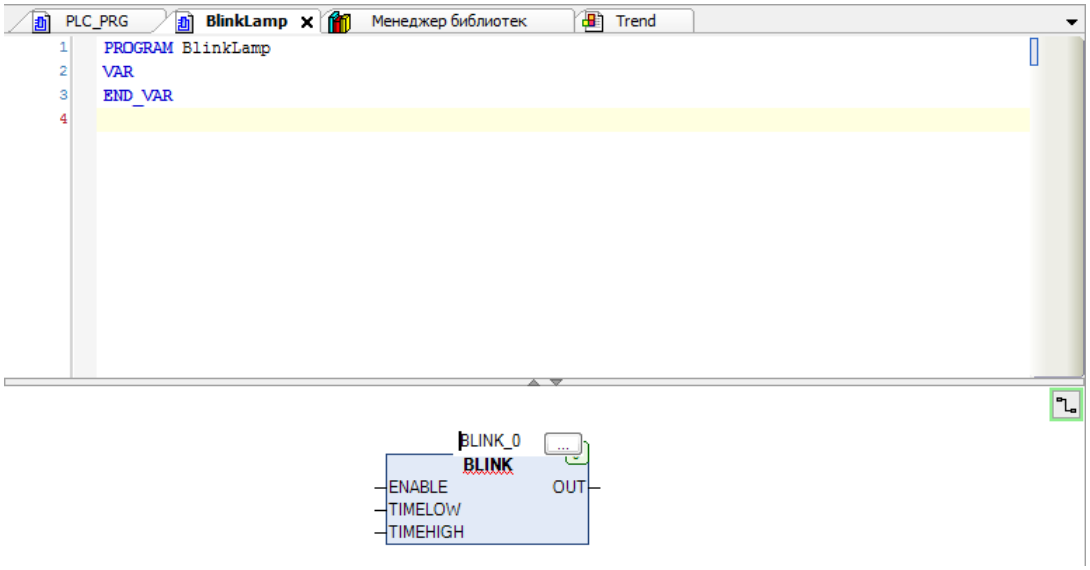


Рис. 7.83. Изменение названия экземпляра функционального блока

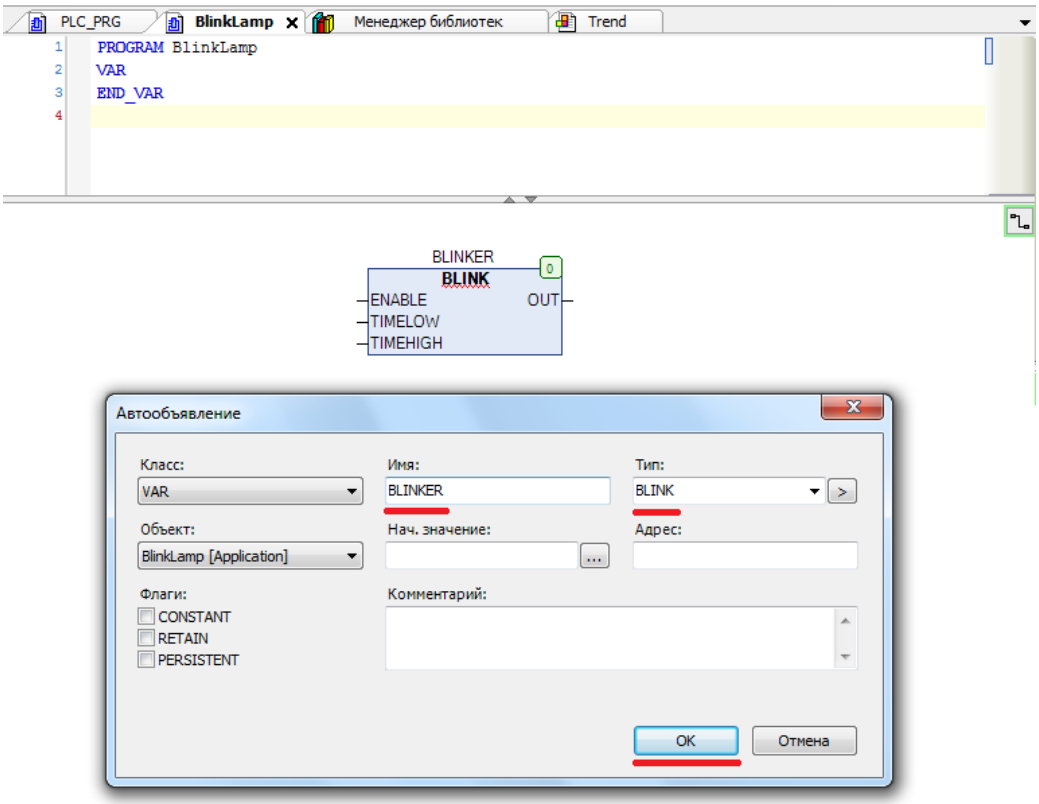


Рис. 7.84. Ассистент автообъявления

После нажатия на **ОК** в **области определения переменных** автоматически создастся экземпляр **BLINKER** функционального блока типа **BLINK**.

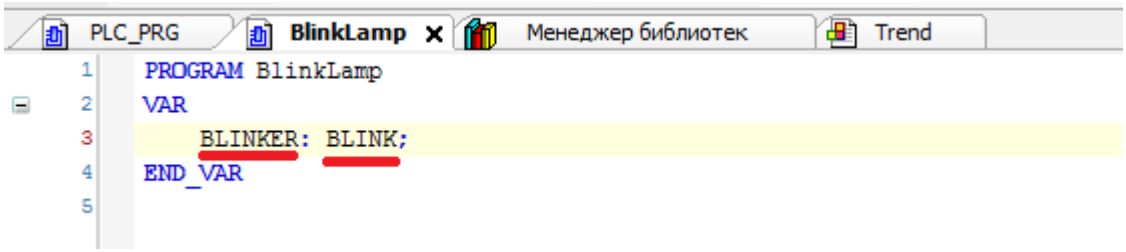


Рис. 7.85. Определение экземпляра функционального блока как переменной

Добавим локальную переменную **yellow_lamp** типа **BOOL**, которая будет характеризовать состояние желтой аварийной лампы на экране тренда: TRUE – лампа горит, FALSE –лампа не горит. Быстрая смена состояний лампы будет визуально соответствовать ее миганию. Остальные переменные, которые мы будем использовать, уже определены в **Списке глобальных переменных** (см. п. 7.4.2).

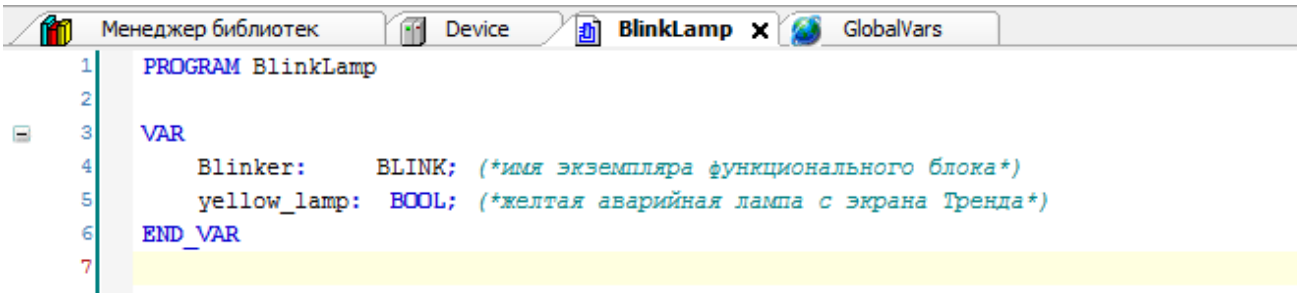


Рис. 7.86. Определение переменных программы **BlinkLamp**

Как можно заметить, функциональный блок BLINK обладает тремя **входами** и одним **выходом**:

Табл. 2. Характеристики переменных блока **BLINK**

Название	Тип	Назначение
Входы		
ENABLE	BOOL	TRUE – запустить BLINK, FALSE – остановить BLINK (на выходе остается текущее значение)
TIMELOW	TIME	Время, которое на выходе блока сохраняется FALSE
TIMEHIGH	TIME	Время, которое на выходе блока остается TRUE
Выходы		
OUT	BOOL	Выход блока (по умолчанию имеет значение FALSE)

Функциональный блок **BLINK** работает следующим образом:

1. При значении ENABLE = TRUE:

    1.1. OUT = FALSE на время TIMELOW;

    1.2. OUT = TRUE на время TIMEHIGH;

    далее последовательно выполняются пункты 1.1. и 1.2.

2. При значении ENABLE = FALSE:

    OUT сохраняет последнее принятое им значение.

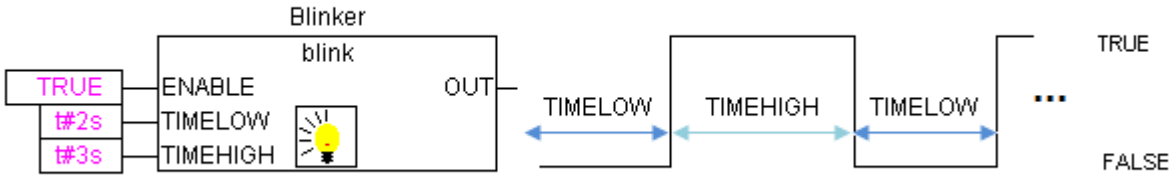


Рис. 7.87. Графическая интерпретация работы BLINK

Напишем программу для мигания лампы в случае выхода температуры за значения аварийных уставок. Если температура будет находиться в пределах уставок, лампа останется потухшей.

Готовая программа будет выглядеть следующим образом:

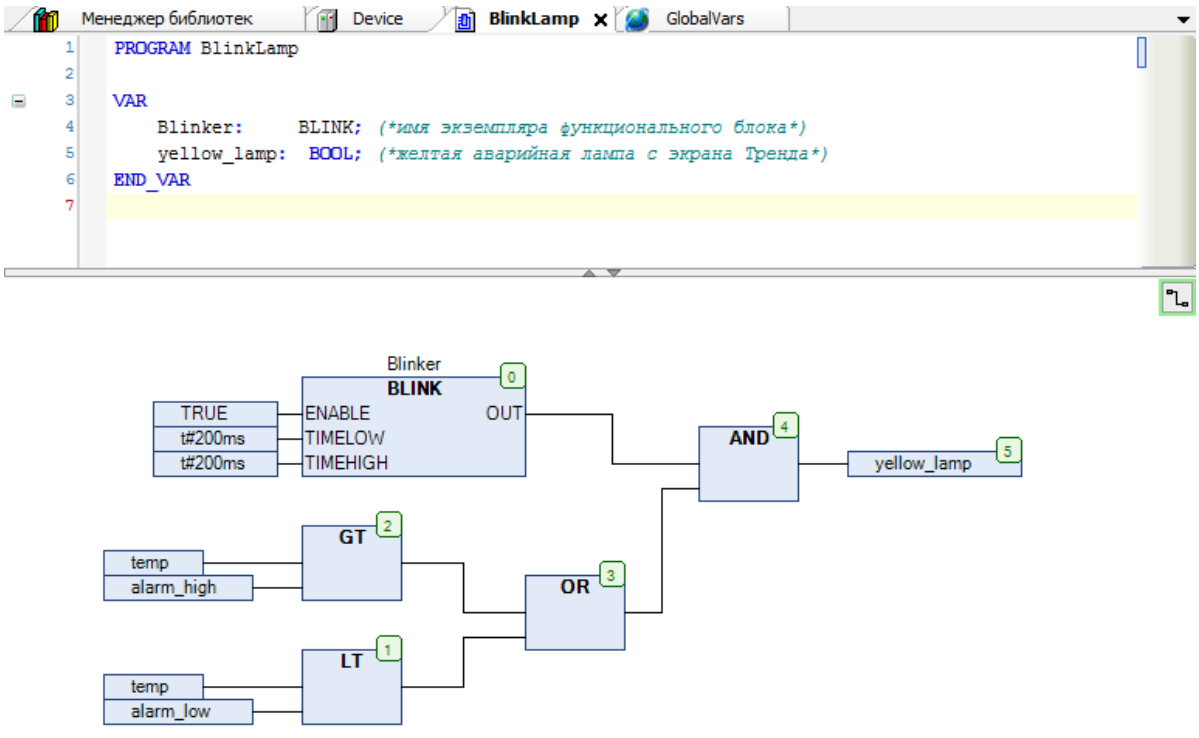


Рис. 7.88. Программа **BlinkLamp**

Блоки **GT**, **LT**, **AND** и **OR** создаются по аналогии с блоком **BLINK**: сначала на **Панели инструментов редактора** нажатием **ЛКМ** выделяется элемент «**Элемент**», затем одиночным нажатием **ЛКМ** размещается в рабочей области, после чего «заполняется» соответствующим функциональным блоком с помощью ввода его названия вручную или же выбора через **Ассистент** (вкладка **Ключевые слова**):

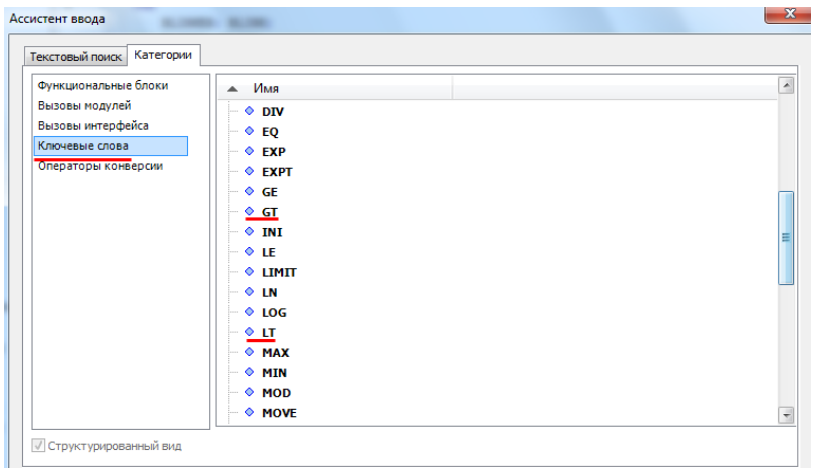


Рис. 7.89. Выбор ключевых слов

Ключевые слова не имеют экземпляров.

GT и **LT** – это операторы сравнения, выходы которых принимают значение TRUE, если значение первого (верхнего) входа больше (**GT**) или меньше (**LT**) значения второго входа. **AND** и **OR** – операторы логического И и ИЛИ соответственно.

Для того, чтобы присвоить входу блока значение, следует выделить его однократным нажатием ЛКМ и начать вводить имя переменной (или константы), что приведет к появлению соответствующей строки с кнопкой **Ассистента ввода**:

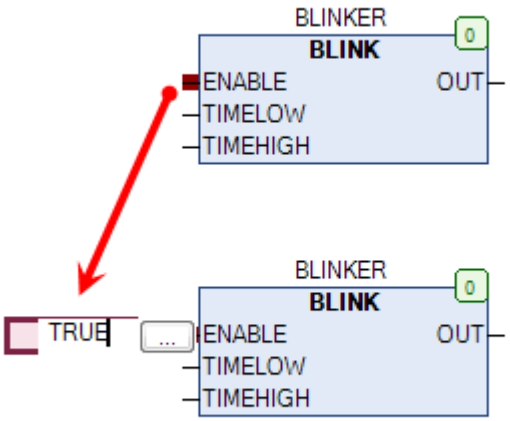


Рис. 7.90. Ввод значений входных переменных функционального блока

Необходимо отметить, что значения входов блока BLINK **TIMELOW** и **TIMEHIGH** для корректной работы должны быть согласованы с **временем цикла** программы, которое определяется соответствующей **задачей** (см. п. 7.7).

Напомним, что **Temp, alarm_low** и **alarm_high** определены в **Списке глобальных переменных** (см. п. 7.4.2).

Для того, чтобы **связать** блоки между собой, необходимо выделить нажатием **ЛКМ** вход/выход первого блока и ***не отпуская*** кнопки мыши перетащить курсор на выход/вход второго.

Значение выхода блока AND присваивается переменной **yellow_lamp** с помощью элемента «**Вывод**».

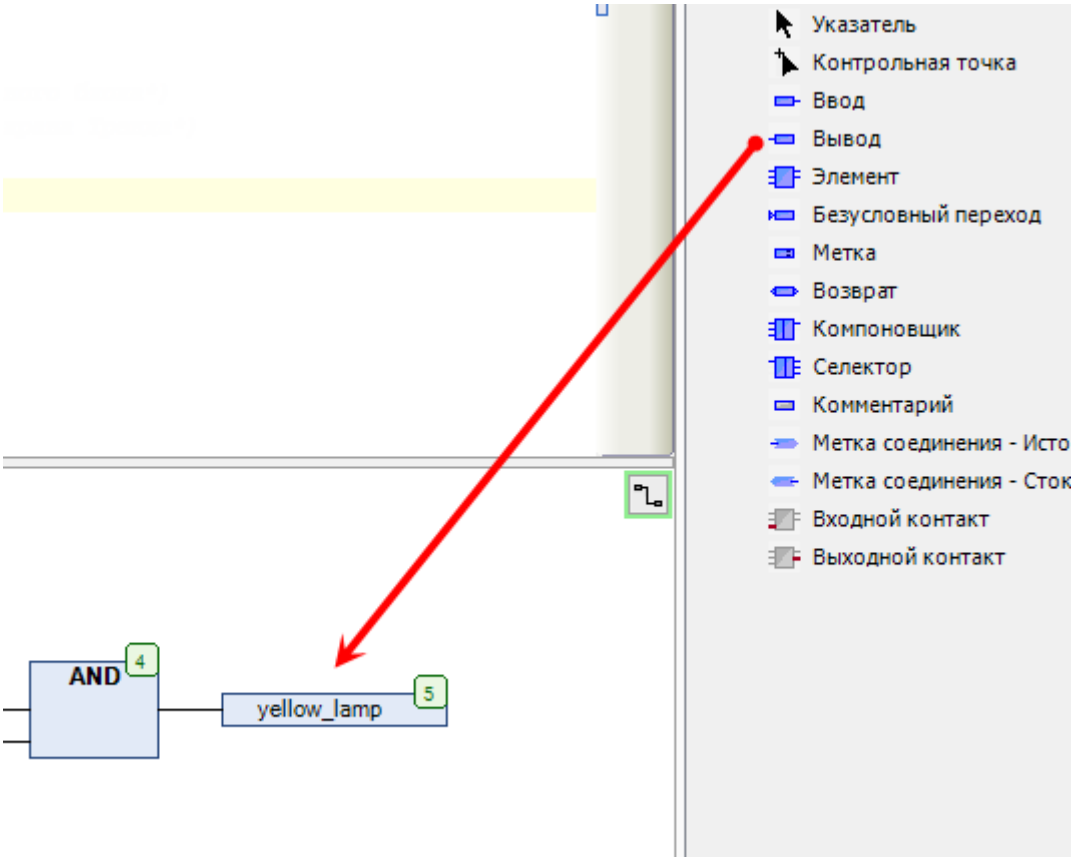


Рис. 7.91. Добавление элемента **Вывод** в редакторе CFC

Программа работает следующим образом: значение на выходе блока **BLINK** (TRUE или FALSE) раз в 200 мс меняется на противоположное. **Выход блока OR** принимает значение TRUE в том случае, если значение температуры выходит за верхнюю или нижнюю аварийную уставку. Если температура находится в допустимых пределах, **выход блока OR** принимает значение FALSE. Соответственно, значение **выхода блока AND**, которое присваивается переменной **yellow_lamp**, однозначно определяется **выходом блока OR**:

1. Если выход OR = TRUE, тогда **yellow_lamp** раз в 200 мс меняет свое значение на противоположное (с FALSE на TRUE или с TRUE на FALSE). Визуально это соответствует **миганию** лампы;
2. Если выход OR = FALSE, тогда **yellow_lamp** = FALSE. Визуально это соответствует **неактивной** (потухшей) лампе.

7.4.4. Функция на языке ST (расчет границ гистерезиса)

Как уже упоминалось ранее, **функция** - это **POU**, при выполнении которого возвращается **только одно** значение. Функции используются для:

1. избавления от необходимости многократно повторять в тексте программы аналогичные фрагменты;

2. улучшения структуры программы, облегчения понимания ее работы;

3. повышения устойчивости к ошибкам программирования и непредвиденным последствиям при модификациях программы.

В нашем примере функция будет использоваться для пересчета значений границ **гистерезиса** из процентов (относительно уставки температуры) в градусы Цельсия.

Создадим **POU** типа **функция** на языке **ST** с названием **temp_hyst**, которое возвращает значение типа **Real**:

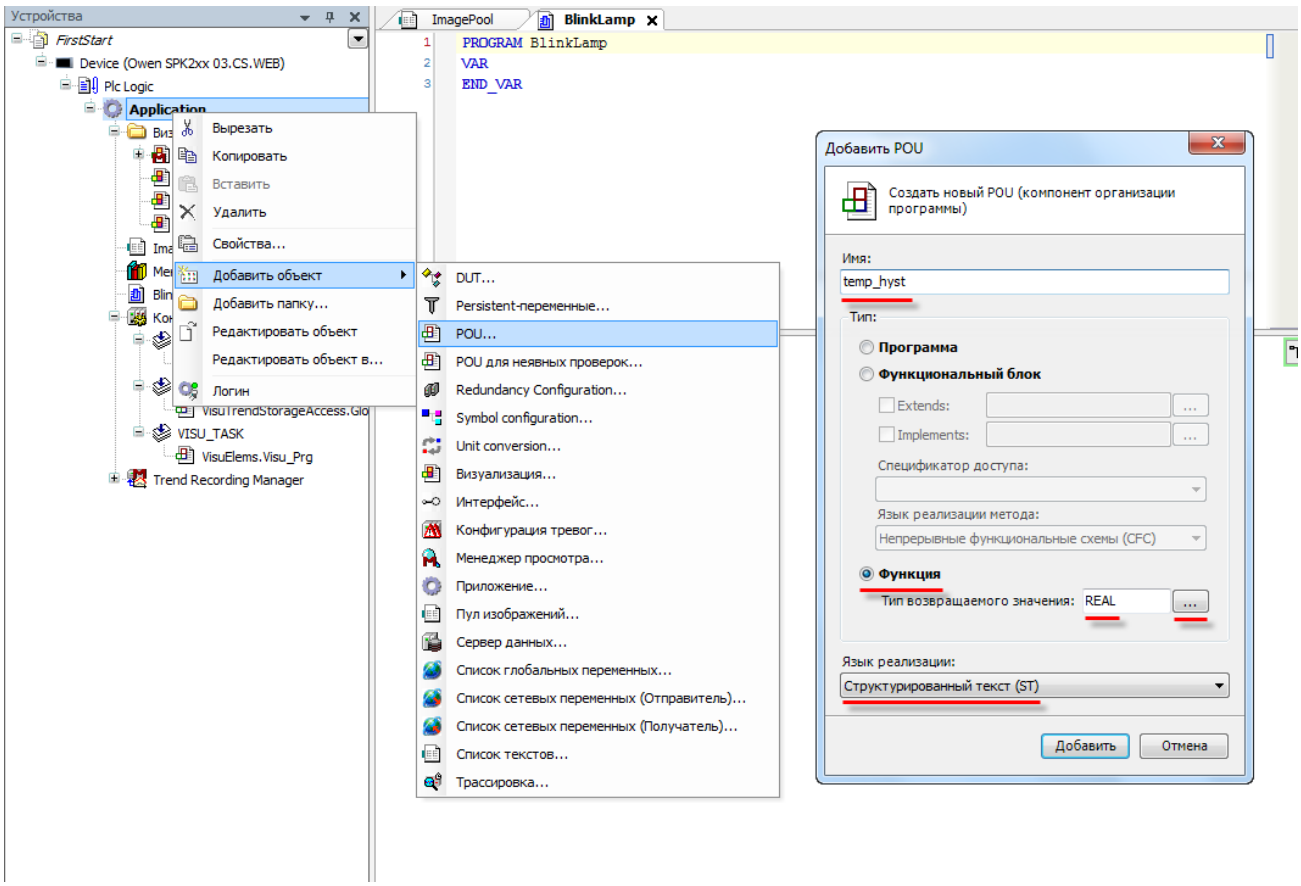


Рис. 7.92. Создание функции

Пользователь задает уставку температуры **temp_ust** в градусах Цельсия и значение гистерезиса в процентах от этой уставки. Это значение будет пересчитано в верхнюю и нижнюю температурные границы гистерезиса, которые соответственно определяются формулами:

$$t_{\text{гист}} = t_{\text{уст}} \pm \frac{hyst}{100} \cdot t_{\text{уст}}$$

Определение переменных и код функции будет выглядеть следующим образом:

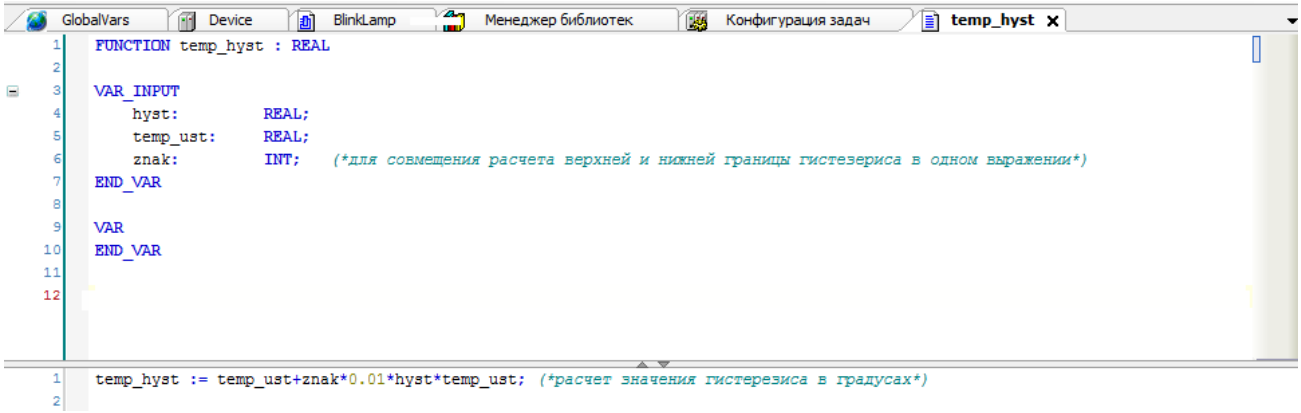


Рис. 7.93. Определение переменных и код функции

Локальные входные переменные функции объявляются между ключевыми словами **VAR_INPUT** и **END_VAR**. ***В данном примере*** при вызове функции (см. п. 7.4.6.) им присваиваются значения **одноименных глобальных переменных**. ***Нужно понимать***, что несмотря на одинаковые названия, это **разные** переменные, относящиеся к **разным видам**: определенные в функции – **локальные**, определенные в Списке глобальных переменных – **глобальные**. При этом в общем случае название локальной переменной ***не обязано соответствовать*** названию переменной, чье значение она принимает. Иными словами, в определении переменных функции мы могли использовать **любые** (отвечающие требованиям CODESYS) названия.

Результат вычисления присваивается **непосредственно самой функции**, поэтому определение выходной переменной не требуется.

Целочисленная переменная **znak** принимает значение «-1» при расчете нижней границы гистерезиса и «+1» – при расчете верхней.

Процедура **вызова функции** будет описана в п. 7.4.6.

7.4.5. Функциональный блок на языке ST (четырехцветная лампа)

Наряду с мигающей лампой, реализация которой рассмотрена в п. 7.4.3, другой типичной задачей является создание **многопозиционного индикатора**, характеризующего состояние какого-либо объекта. В самом простом случае таким индикатором является **аварийная лампа** с двумя состояниями – **Авария** (красный цвет лампы) и **Норма** (зеленый цвет лампы).

Среди **графических примитивов** CODESYS присутствуют только **двухпозиционные** лампы, причем оба состояния каждой из них ***жестко фиксированы***: TRUE – свечение соответствующего оттенка, FALSE – неактивное состояние (лампа не горит). Для того, чтобы создать многопозиционную лампу, нам потребуется в **Редакторе визуализации** наложить несколько ламп друг на друга, а так же подготовить программу, переключающую их состояния.

В нашем примере цвет многопозиционной лампы будет отражать **состояние кондиционера** – включен или отключен, причем если кондиционер включен, то цвет лампы будет характеризовать **режим его работы**. Принцип действия лампы будет описан в **функциональном блоке** на языке **ST**. Напомним, что в п. 7.4.3 мы уже использовали готовый функциональный блок. Теперь же мы создадим подобный элемент самостоятельно.

Создадим **функциональный блок** на языке **ST** с названием **Lamp**. Параметры Extentds, Implements и Спецификатор доступа используются при **объектно-ориентированном подходе** к написанию программ, который в данном примере ***не рассматривается.***

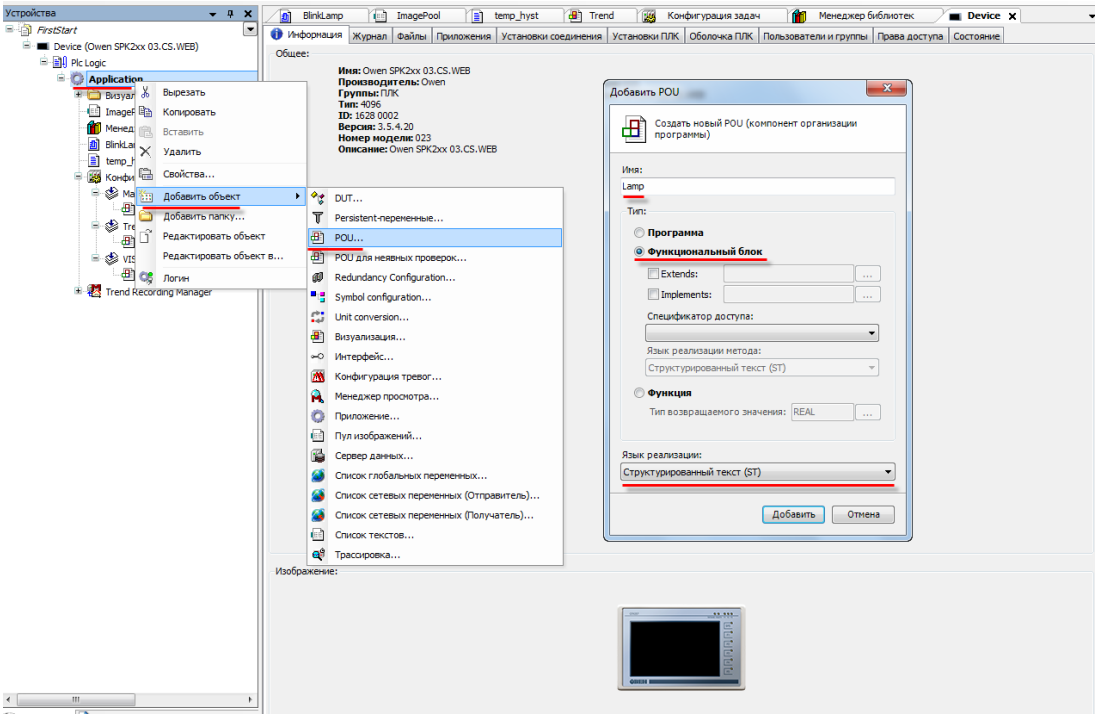


Рис. 7.94. Создание функционального блока

Для начала следует определиться с принципом работы лампы. Ее цвет будет зависеть от **состояния кондиционера** (вкл./откл.), **режима работы регулятора** (вкл./откл.) и **режима работы кондиционера** (нагрев/охлаждение), которые соответственно определяются глобальными переменными **conditioner_power, regulator_state** и **control_mode**. Таблица возможных сочетаний и соответствующих им цветов лампы приведена ниже:

Табл. 3. Таблица состояний лампы

Состояние переменной conditioner_power	**Состояние переменной regulator_state**	**Состояние переменной control_mode**	**Цвет лампы**
TRUE	TRUE	FALSE	
TRUE	TRUE	TRUE	
TRUE	FALSE	не важно	
FALSE	не важно	не важно	

Соответственно, **локальными входными переменными** нашего функционального блока будут являться **conditioner_power_fb**, **regulator_state_fb и control_mode_fb**, которым при вызове блока будут присваиваться значения соответствующих **глобальных** переменных **conditioner_power**, **regulator_state** и **control_mode**. Выходами блока являются **семь** локальных переменных, описывающих состояние **четырех** ламп (одна переменная отвечает за свечение лампы, вторая – за невидимость; неактивной серой лампе не нужна переменная свечения). Выходные переменные определяются между ключевыми словами **VAR_OUTPUT** и **END_VAR**.

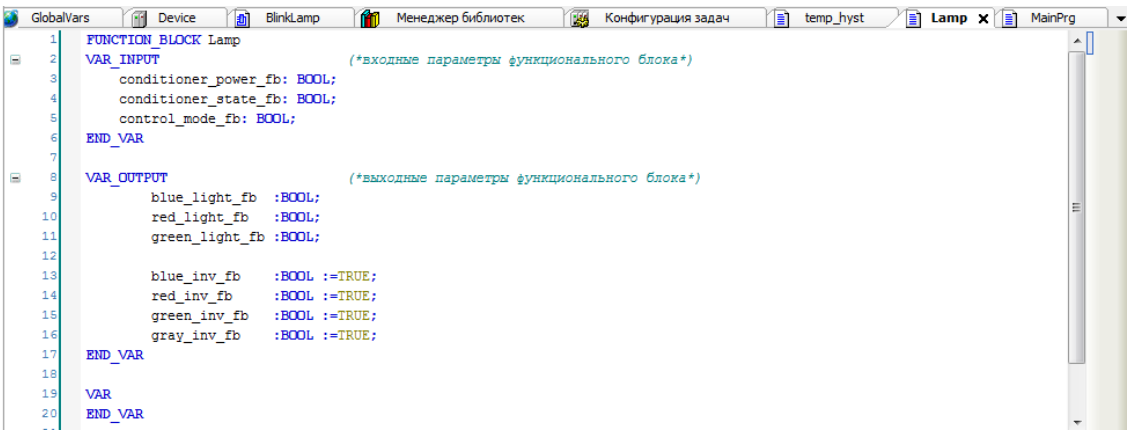


Рис. 7.95. Определение переменных функционального блока

Код программы будет выглядеть следующим образом:

```
(*цвет горящей лампы в зависимости от состояния кондиционера, состояния регулятора и режима работы кондиционера*)

IF conditioner_power = TRUE AND regulator_state = TRUE AND control_mode = FALSE THEN
        blue_light_fb  := TRUE;
        blue_inv_fb    := FALSE;

        red_inv_fb     := TRUE;
        green_inv_fb   := TRUE;
        gray_inv_fb    := TRUE;
    ELSIF conditioner_power = TRUE AND regulator_state = TRUE AND control_mode = TRUE THEN
        red_light_fb   := TRUE;
        red_inv_fb     := FALSE;

        blue_inv_fb    := TRUE;
        green_inv_fb   := TRUE;
        gray_inv_fb    := TRUE;
    ELSIF conditioner_power = TRUE AND regulator_state = FALSE THEN
        green_light_fb := TRUE;
        green_inv_fb   := FALSE;

        blue_inv_fb    := TRUE;
        red_inv_fb     := TRUE;
        gray_inv_fb    := TRUE;
    ELSIF conditioner_power = FALSE THEN
        blue_inv_fb    := TRUE;
        red_inv_fb     := TRUE;
        green_inv_fb   := TRUE;
        gray_inv_fb    := FALSE;
END_IF
```

Рис. 7.96. Код функционального блока

Синтаксис **конструкции с оператором IF** следующий (форматирование использовано исключительно для наглядности):

IF <Логическое условие 1> THEN <Действие 1>
{ELSIF <Логическое условие 2> THEN <Действие 2>
...
ELSIF <Логическое условие n> THEN <Действие n>
ELSE <Действие n+1>}
END_IF;

Использование операторов **ELSIF** и **ELSE** опционально.

Конструкция работает следующим образом:

1. Если Логическое условие 1 возвращает значение TRUE, то выполняется Действие 1.
2. Если Логическое условие 1 возвращает значение FALSE, то осуществляется переход к оператору ELSIF с Логическим условием 2, который работает аналогично оператору IF.
3. Если все n Логических условий вернули FALSE, то осуществляется переход к оператору ELSE и выполняется соответствующее Действие n+1.

Итак, для реализации четырехпозиционной лампы мы наложим четыре лампы различных цветов друг на друга и привяжем к ним выходы функционального блока. В результате, ***в каждый момент времени будет видна только одна*** лампа (какая – определяется значениями входных переменных), а остальные будут ***невидимы***.

Процедура **вызова функционального блока** будет описана в п. 7.4.6.

7.4.6. Программа на языке ST (регулятор и модель объекта)

Итак, в предыдущих пунктах мы создали **функцию** и **функциональный блок**. Но для реализации их работы необходимо создать **программу**, в которой они будут вызываться (см. рис. 7.73). Создадим **POU** типа **программа** на языке **ST** с названием **MainPrg:**

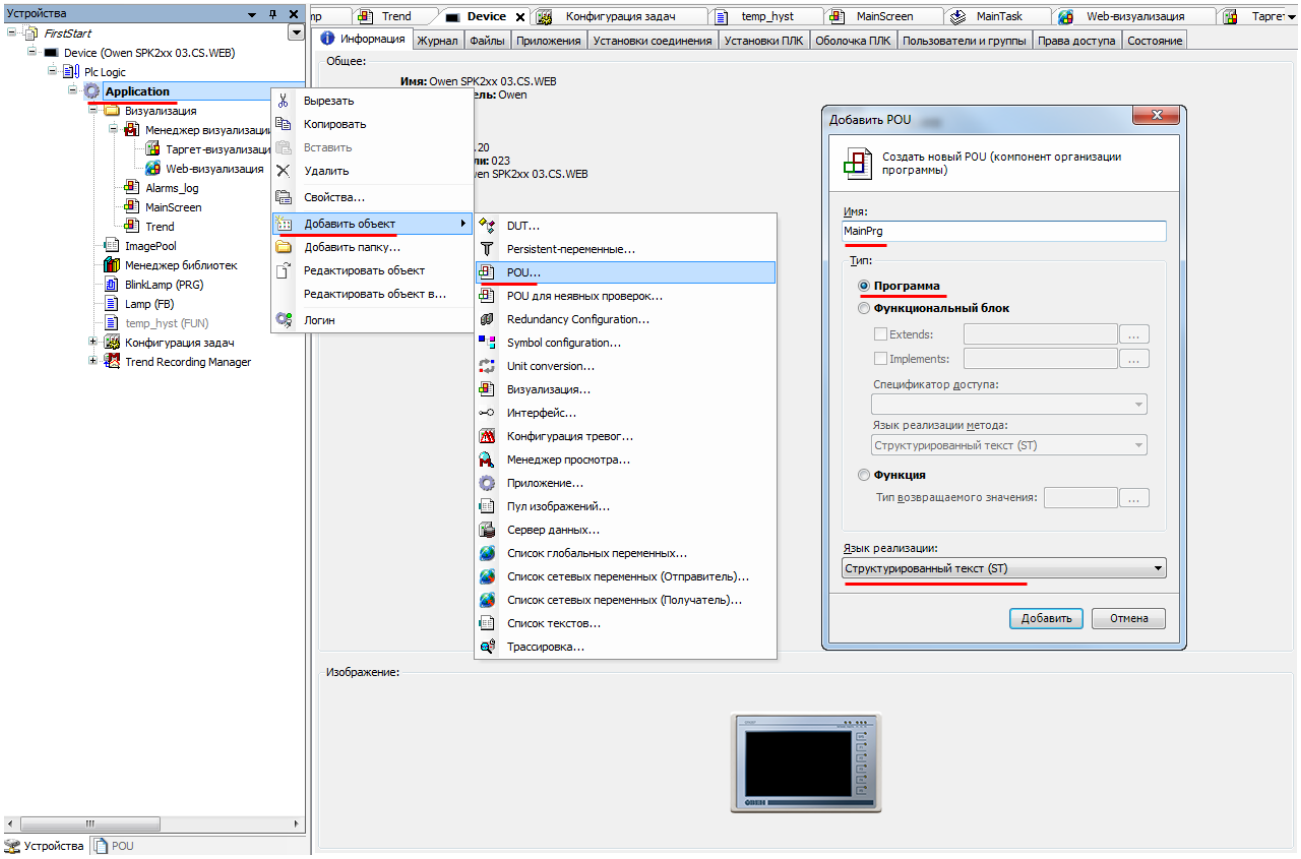


Рис. 7.97. Создание программы **MainPrg**

Программа будет выполнять **следующие действия**:

1. Определять текущий режим измерения (измерение с датчика или режим эмуляции измерения);

2. С помощью вызова функции **temp_hyst** рассчитывать значения границ гистерезиса;

3. На основании положения значения температуры относительно границ гистерезиса переключать кондиционер в соответствующий режим;

4. При задании температуры пользователем эмулировать процесс изменения температуры при работе кондиционера;

5. С помощью вызова функционального блока **Lamp** подсвечивать индикатор кондиционера цветом, соответствующим режиму его работы.

Определим локальные переменные программы:

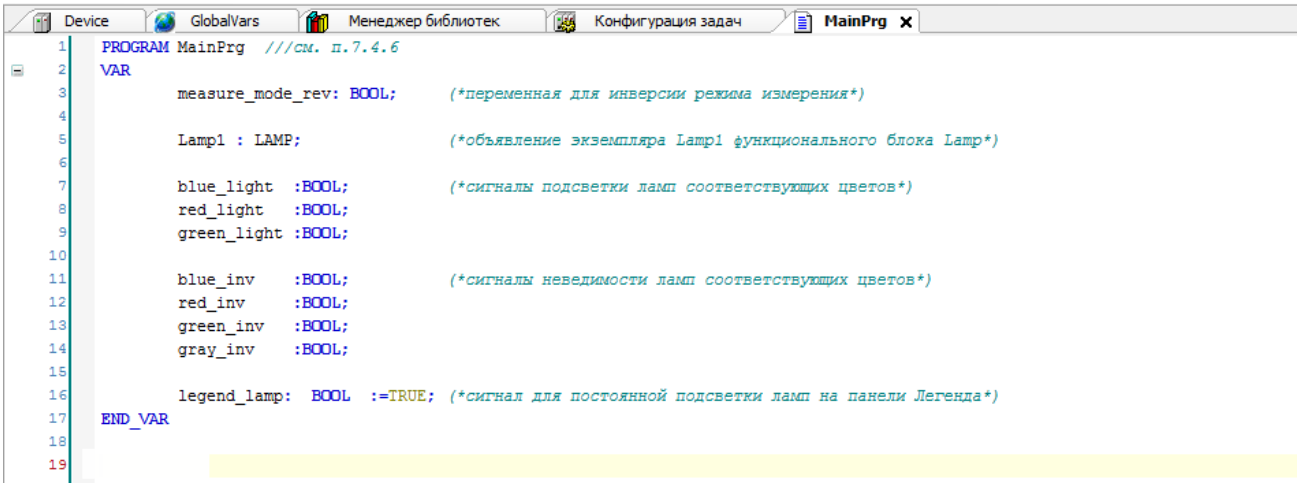


Рис. 7.98. Определение переменных программы

Все переменные, за исключением **measure_mode_rev**, используются для реализации **четырехпозиционной лампы**.

Логическая переменная **measure_mode_rev** используется для **инверсии** значения глобальной переменной **measure_mode**:

```
measure_mode_rev := NOT measure_mode;
```

Соответственно, она принимает значение TRUE, когда measure_mode = FALSE и наоборот.

Эта переменная используется для ***скрытия кнопок управления*** температурой при получении ее значения с датчика. Для того, чтобы кнопки стали **невидимыми**, переменная состояния, привязанная к ним, должна принять значение TRUE. Но значение TRUE переменной **measure_mode** соответствует **режиму ручного задания температуры**, и поэтому данная переменная ***не может быть использована*** как переменная состояния. По этой причине мы создаем **дополнительную** инвертированную переменную.

Далее рассмотрим пошаговый процесс создания программы в соответствии с ***описанной выше*** последовательностью выполняемых ею действий.

I. Определение режима измерения

```
5
6
7   ////////////////////  п.7.4.6, действие 1  ////////////////////
8
9   (*определение режима измерения*)
10  IF          measure_mode = FALSE THEN temp := temp_real;
11      ELSIF   measure_mode = TRUE  THEN temp := temp_user;
12  END_IF
13
14  ////////////////////////////////////////////////////////////////
15
```

Рис. 7.99. Код действия I

Режим измерения (показания с датчика или эмуляция измерения) определяется пользователем с помощью **чекбокса** на экране **MainScreen**. При наличии галочки пользователь задает значение температуры ***вручную***, а в контроллере происходит **эмуляция** работы кондиционера; при отсутствии – используется значение температуры **с датчика**. Каждому из режимов измерения соответствует своя переменная температуры (**temp_real** и **temp_user**). Значение переменной текущего режима присваивается переменной **temp** для дальнейшего использования в коде программы с помощью конструкции **IF**, рассмотренной в п. 7.4.5.

II. Расчет границ гистерезиса. Вызов функции

```
17
18  ////////////////////  п.7.4.6, действие 2  ////////////////////
19
20  (*вызов функции temp_hyst для пересчета гистерезиса из процентов в температуру*)
21  temp_hyst_low  := temp_hyst(hyst,temp_ust,-1);
22  temp_hyst_high := temp_hyst(hyst,temp_ust,1);
23
24  ////////////////////////////////////////////////////////////////
25
```

Рис. 7.100. Код действия II

Пользователь задает уставку температуры **temp_ust** в градусах Цельсия и значение гистерезиса в процентах от этой уставки. Для того, чтобы перевести значения границ **гистерезиса** из процентов в градусы Цельсия, используется **функция temp_hyst**, рассмотренная в п. 7.4.4.

Синтаксис вызова функции следующий:

<Имя переменной> := <Имя функции> (<аргумент функции 1>, …, <аргумент функции n>);

где

<**Имя переменной**> - имя переменной, которой присваивается результат вызова функции. Тип переменной должен ***совпадать*** с типом функции, иначе возможна потеря данных или некорректная работа программы;

<**Имя функции**> - название функции;

<**аргумент функции**> - переменная, значение которой присваивается входной переменной функции. Число аргументов должно ***строго соответствовать*** числу входных переменных, а их порядок – последовательности объявления переменных функции.

III. Определение режима работы кондиционера

```
28  ////////////////////  п.7.4.6, действие 3  ////////////////////
29
30  (*выбор режима работы кондиционера*)
31  IF        temp < temp_hyst_low  AND conditioner_power  = TRUE THEN
32                                      control_mode      := TRUE;
33                                      regulator_state   := TRUE;
34  (*температура ниже гистерезиса--->включаем подогрев*)
35      ELSIF temp > temp_hyst_high AND conditioner_power  = TRUE THEN
36                                      control_mode      := FALSE;
37                                      regulator_state   := TRUE;
38  (*температура выше гистерезиса--->включаем охлаждение*)
39      ELSIF temp >= temp_hyst_low AND  temp <= temp_hyst_high   THEN regulator_state := FALSE;
40  (*температура в пределах гистерезиса--->переводим кондиционер в режим ожидания*)
41  END_IF
42
43  ////////////////////////////////////////////////////////////
```

Рис. 7.101. Код действия III

Режим работы кондиционера определяется положением температуры относительно границ гистерезиса, а также состояниями кондиционера и регулятора (вкл./откл.). Очевидно, что если кондиционер выключен, говорить о режиме его работы не имеет смысла. Если же он включен, то может работать в режиме нагрева/охлаждения (при условии, что включен регулятор и температура выходит за нижнюю/верхнюю уставку гистерезиса) или режиме ожидания (регулятор включен, но температура находится в допустимых пределах). Это реализуется с помощью **конструкции IF**. От приведенных ранее примеров данная вариация конструкции отличается лишь тем, что в ее условии используется **оператор AND** (логическое И). ***Следует отметить***, что этот оператор ***нельзя*** использовать в **действиях**, т.е. запись типа

```
IF <Логическое условие 1>   THEN   <Действие 1>  AND <Действие 2> ;
END_IF;
```

некорректна.

Если **конструкция IF** подразумевает исполнение **нескольких** действий в результате выполнения логического условия, то они перечисляются через пробел, причем каждое из них заканчивается точкой с запятой (**;**).

IV. Эмуляция изменения температуры

```
47  ////////////////////  п.7.4.6, действие 4  ////////////////////
48
49  IF        conditioner_power = TRUE AND regulator_state = TRUE AND control_mode = FALSE AND measure_mode = TRUE THEN
50      temp := temp-3;
51  (*работа кондиционера в режиме охлаждения*)
52      ELSIF conditioner_power = TRUE AND regulator_state = TRUE AND control_mode = TRUE  AND measure_mode = TRUE THEN
53      temp := temp+3;
54  (*работа кондиционера в режиме подогрева*)
55  END_IF
56
57
58  IF          measure_mode = FALSE THEN temp_real := temp;  (*запись температуры после _работы кондиционера в данном цикле_...*)
59      ELSIF   measure_mode = TRUE  THEN temp_user := temp;  (*...для возможности переключения режима измерения в ходе цикла *)
60  END_IF
61
62  ////////////////////////////////////////////////////////////
63
64
```

Рис. 7.102. Код действия IV

В случае отсутствия реальных входных и выходных сигналов программа должна работать в **режиме эмуляции**. Эмуляция входного сигнала осуществляется вводом пользователем текущего значения температуры при условии, что выбран соответствующий режим измерения (**measure_mode** = TRUE). Для эмуляции выходных сигналов необходимо реализовать изменение значения температуры при работе кондиционера в режиме нагрева или охлаждения. Это отражено в приведенном выше коде. **Число «3»** является «***мощностью***» кондиционера – оно означает, что в случае, если кондиционер работает в режиме нагрева или охлаждения, при каждом выполнении программы температура в помещении увеличивается или уменьшается на **3 градуса Цельсия** соответственно. После этого ее новое значение присваивается температурной переменной режима измерения. Это необходимо для реализации возможности переключения режима измерения.

V. Подсветка лампы. Вызов функционального блока

```
66  ////////////////////  п.7.4.6, действие 5  ////////////////////
67
68  Lamp1 (*вызов экземпляра Lamp1 функционального блока Lamp*)
69      (
70      conditioner_power_fb := conditioner_power,
71      conditioner_state_fb := regulator_state,
72      control_mode_fb      := control_mode,
73
74      blue_light_fb  => blue_light,
75      red_light_fb   => red_light,
76      green_light_fb => green_light,
77
78      blue_inv_fb    => blue_inv,
79      red_inv_fb     => red_inv,
80      green_inv_fb   => green_inv,
81      gray_inv_fb    => gray_inv,
82      );
83
84  ////////////////////////////////////////////////////////////////
```

Рис. 7.103. Код действия V

Для реализации **четырехпозиционной лампы**, цвет которой зависит от режима работы кондиционера (см. табл. 3), мы используем **функциональный блок Lamp**, созданный в п. 7.4.5 (там же описан и принцип работы лампы). Напоминаем, что **вызов функционального блока** осуществляется через обращение к его **экземпляру**.

Пример синтаксиса вызова функционального блока следующий:

Block_ex (in1: = a, in2 := b, out => c);

где

<**Block_ex**> - имя экземпляра функционального блока;

<**in1**>, <**in1**> - имена входных переменных функционального блока;

<**a**>, <**b**> - имена переменных, чьи значения присваиваются входным переменным функционального блока;

<**out**> - имя выходной переменной функционального блока;

<**c**> - имя переменной, которой присваивается выходное значение функционального блока (***обратите внимание*** на специальный оператор присваивания **=>**).

В приведенном выше коде каждой входной переменной функционального блока при его вызове присваивается значение локальной переменной программы, а значение каждой выходной переменной функционального блока в свою очередь присваивается соответствующей локальной переменной программы. Это **не обязательное** требование; функциональный блок можно вызывать и **без обращения** ко всем его входам/выходам, и даже без обращения к ним вовсе, т.е. в определенных случаях запись типа

```
Block_ex();
```

является легитимной.

Существует также **альтернативный вариант** синтаксиса вызова функционального блока (с поочередным обращениям к его входам/выходам):

```
Block_ex.in1 :=a;
Block_ex.in2 :=b;
Block_ex();              (*вызов блока *);
c :=Block_ex.out ;
```

Поскольку теперь у нас появилось несколько однотипных программных компонентов , с помощью нажатия **ПКМ** на компонент **Applicaton** на **Панели устройств** создадим новую папку **Программы**, и, ***зажав* ЛКМ**, перенесем туда созданные нами программы:

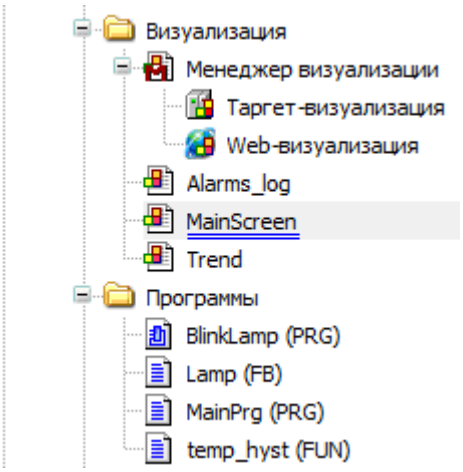


Рис. 7.104. Папка **Программы** на **Панели устройств**

Полный код программы **MainPrg** выглядит следующим образом:

```
(*сигнал для скрытия кнопок изменения текущей температуры при получении значения температуры с датчика*)
measure_mode_rev := NOT measure_mode;


//////////////////  п.7.4.6, действие 1  //////////////////

(*определение режима измерения*)
IF          measure_mode = FALSE THEN temp := temp_real;
    ELSIF   measure_mode = TRUE  THEN temp := temp_user;
END_IF

////////////////////////////////////////////////////////////


//////////////////  п.7.4.6, действие 2  //////////////////

(*вызов функции temp_hyst для пересчета гистерезиса из процентов в температуру*)
temp_hyst_low  := temp_hyst(hyst,temp_ust,-1);
temp_hyst_high := temp_hyst(hyst,temp_ust,1);

////////////////////////////////////////////////////////////


//////////////////  п.7.4.6, действие 3  //////////////////

(*выбор режима работы кондиционера*)
IF        temp < temp_hyst_low  AND conditioner_power  = TRUE THEN
                                    control_mode      := TRUE;
                                    regulator_state   := TRUE;
(*температура ниже гистерезиса--->включаем подогрев*)
    ELSIF temp > temp_hyst_high AND conditioner_power  = TRUE THEN
                                    control_mode      := FALSE;
                                    regulator_state   := TRUE;
(*температура выше гистерезиса--->включаем охлаждение*)
    ELSIF temp >= temp_hyst_low AND  temp <= temp_hyst_high   THEN regulator_state := FALSE;
(*температура в пределах гистерезиса--->переводим кондиционер в режим ожидания*)
END_IF

////////////////////////////////////////////////////////////


//////////////////  п.7.4.6, действие 4  //////////////////

IF        conditioner_power = TRUE AND regulator_state = TRUE AND control_mode = FALSE AND measure_mode = TRUE THEN
    temp := temp-3;
(*работа кондиционера в режиме охлаждения*)
    ELSIF conditioner_power = TRUE AND regulator_state = TRUE AND control_mode = TRUE  AND measure_mode = TRUE THEN
    temp := temp+3;
(*работа кондиционера в режиме подогрева*)
END_IF


IF          measure_mode = FALSE THEN temp_real := temp;  (*запись температуры после _работы кондиционера в данном цикле_...*)
    ELSIF   measure_mode = TRUE  THEN temp_user := temp;  (*...для возможности переключения режима измерения в ходе цикла *)
END_IF

////////////////////////////////////////////////////////////


//////////////////  п.7.4.6, действие 5  //////////////////

Lamp1 (*вызов экземпляра Lamp1 функционального блока Lamp*)
    (
    conditioner_power_fb := conditioner_power,
    conditioner_state_fb := regulator_state,
    control_mode_fb      := control_mode,

    blue_light_fb  => blue_light,
    red_light_fb   => red_light,
    green_light_fb => green_light,

    blue_inv_fb    => blue_inv,
    red_inv_fb     => red_inv,
    green_inv_fb   => green_inv,
    gray_inv_fb    => gray_inv,
    );

////////////////////////////////////////////////////////////
```

Рис. 7.105. Полный код программы MainPrg

7.5. Связь визуализации и программных переменных. Настройка кнопок

Итак, мы подготовили визуальную и программную части нашего приложения – первая включает в себя **три экрана визуализации**, вторая - **две программы** на языках ST и CFC, а так же **функцию** и **функциональный блок**, вызываемые ST программой. Теперь необходимо осуществить **привязку переменных** программ к элементам визуализации, а также настроить **действия** кнопок.

7.5.1. Экран MainScreen

I. Привязка переменных

Для начала привяжем к элементу **Отображение линейки** переменную **temp**, которая характеризует текущее значение температуры. Для этого выделим элемент и дважды нажмем **ЛКМ** на параметр **Значение** в его свойствах:

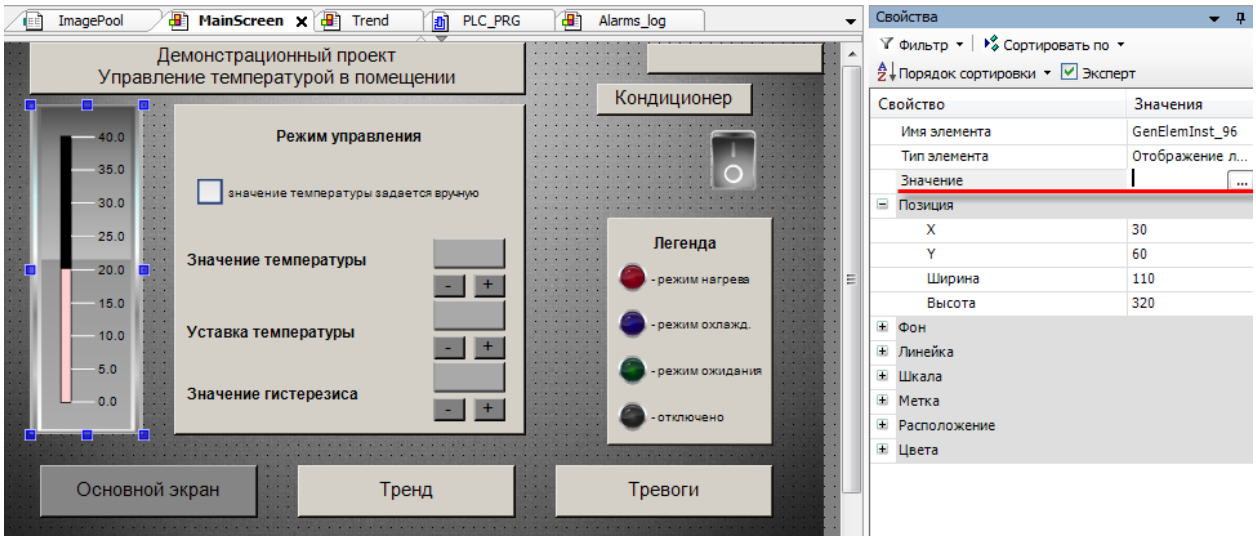


Рис. 7.106. Параметр **Значение** в свойствах элемента

После этого можно либо ввести название переменной вручную, либо с помощью нажатия соответствующей кнопки выбрать ее через **Ассистент ввода**:

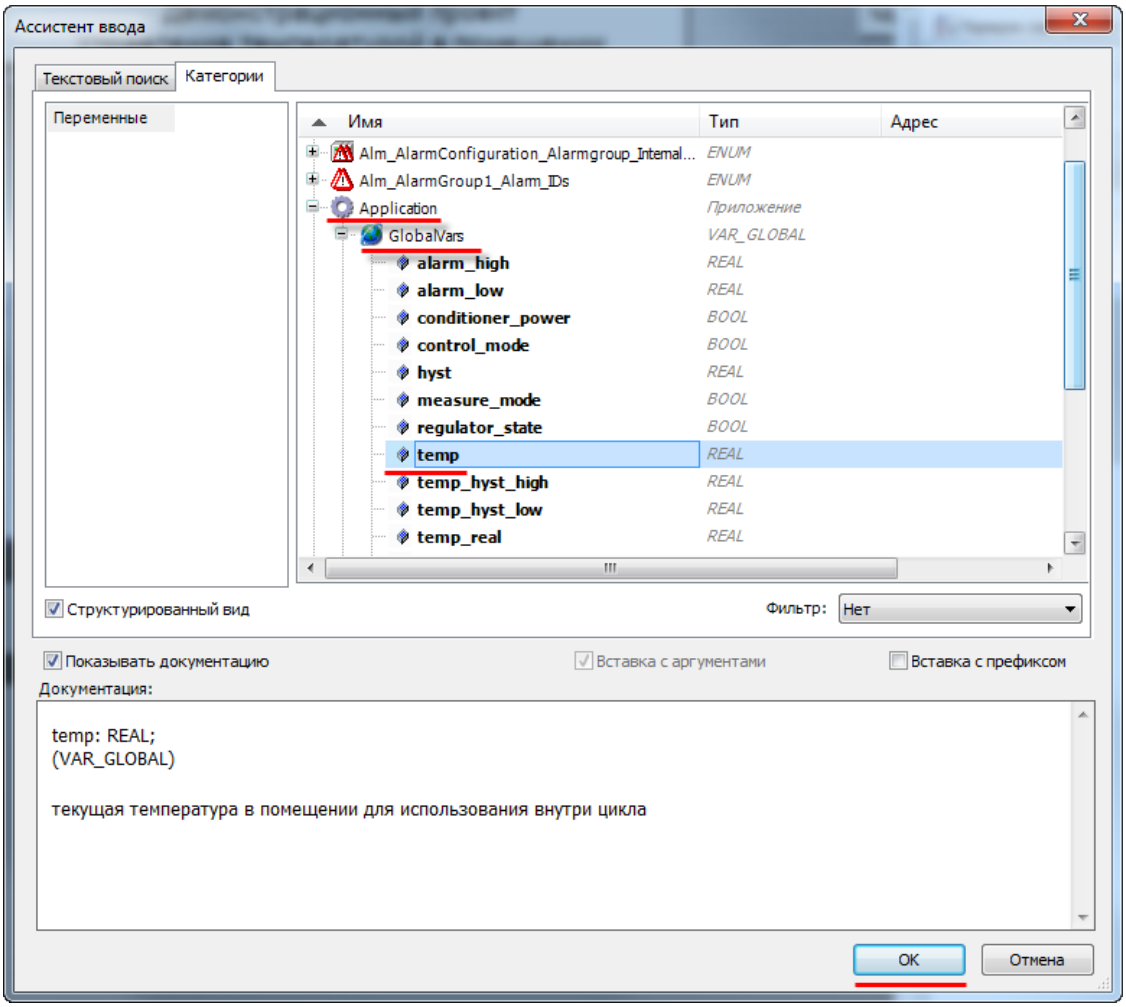


Рис. 7.107. Ассистент привязки переменной

На этом привязка переменной к данному графическому элементу завершена.

Привязка других элементов осуществляется аналогично, за исключением того, что параметр **Значение** у некоторых элементов может называться **Переменная**.

Привяжем к элементу **Чекбокс** переменную **measure_mode** (выбор режима измерения), а к **Клавишному выключателю** – **conditioner_power** (управление питанием выключателя).

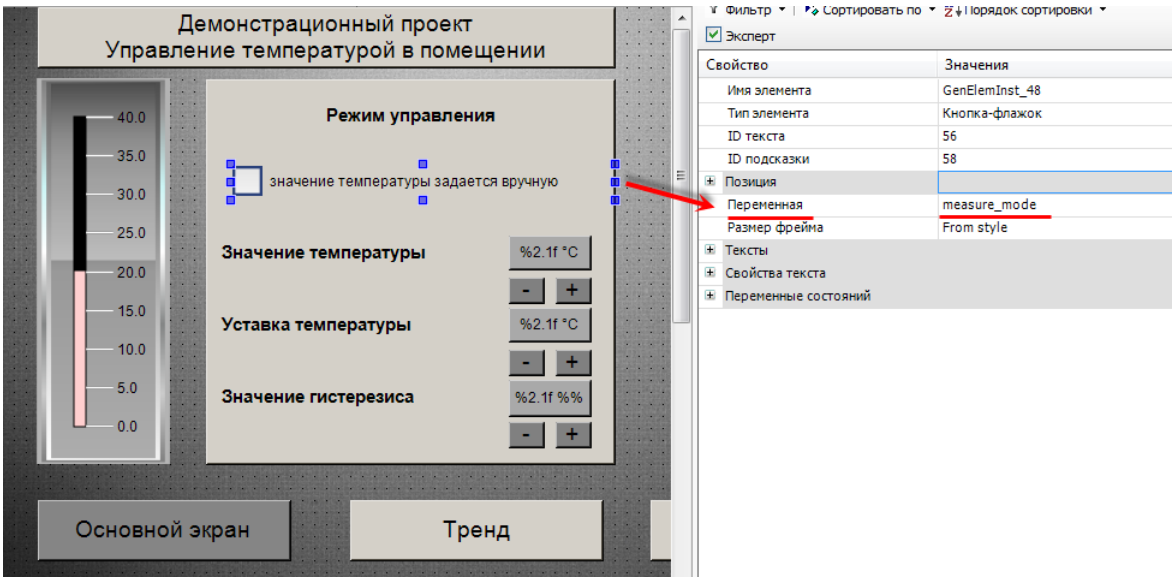


Рис. 7.108. Привязка переменной **measure_mode** к чекбоксу

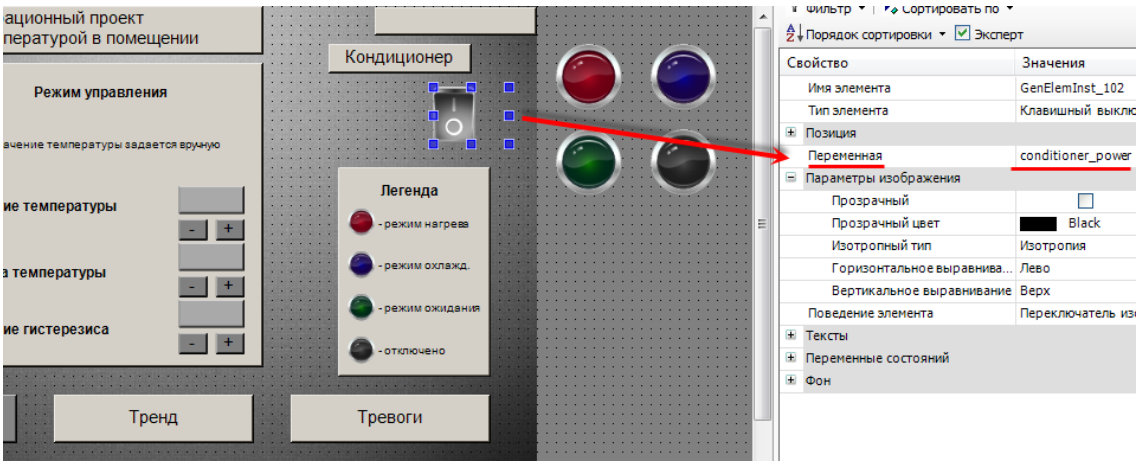


Рис. 7.109. Привязка переменной **conditioner_power** к выключателю

Теперь приступим к привязке переменных к индикаторам, пока что расположенных нами за пределами рабочего поля. К «цветным» индикаторам будет привязано по две переменных – одна определяет состояние индикатора (святящийся/потухший), вторая – видимость (невидимый/видимый). Напомним, что это необходимо для реализации многопозиционного индикатора (см. п. 7.4.5).

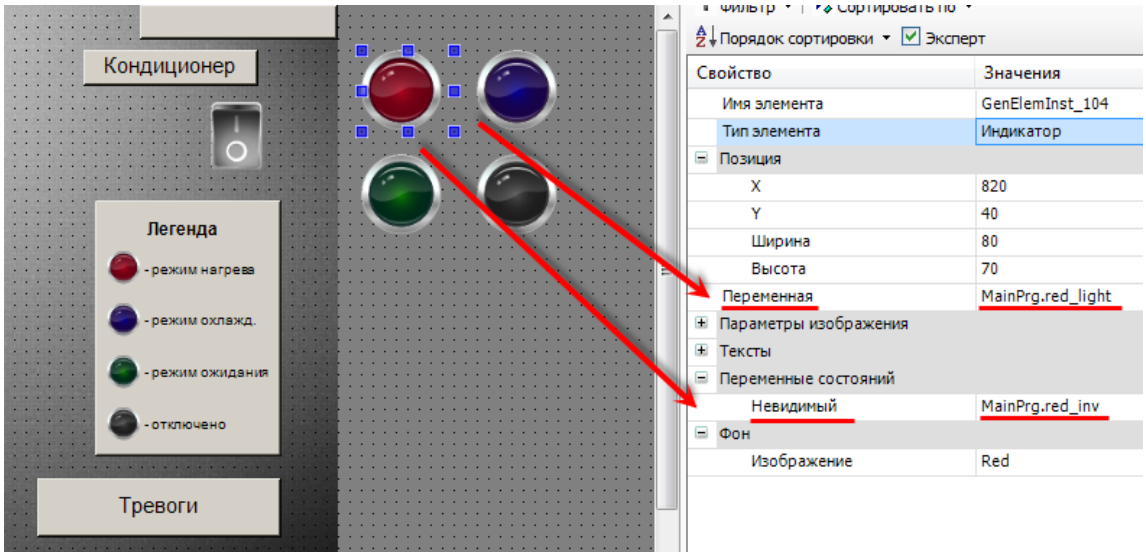


Рис. 7.110. Привязка переменных к индикатору red_lamp

Обратите внимание, что в отличие от предыдущих, переменные **red_light** и **red_inv** являются ***локальными*** переменными программы **MainPrg**, поэтому при наборе их названий вручную необходимо указывать также название программы. При добавлении переменных через **Ассистент ввода** правильное название формируется автоматически.

Аналогичным образом привяжем переменные к остальным индикаторам (см. табл. 4). К серому индикатору будет привязана только переменная **невидимости**, т.к. лампа никогда не будет подсвечиваться серым цветом. Теперь наложим эти лампы друг на друга и поместим получившийся **многопозиционный индикатор** рядом с выключателем кондиционера:

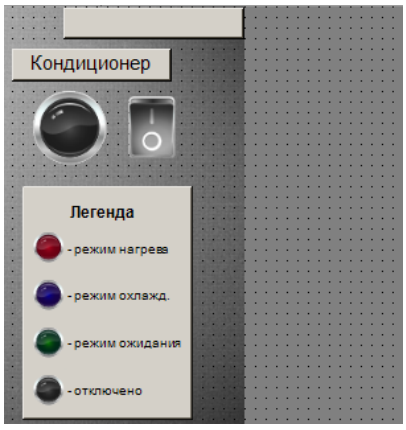


Рис. 7.111. Многопозиционный индикатор, полученный наложением четырех индикаторов друг на друга

К «цветным» индикаторам на панели **Легенда** привяжем переменную **legend_lamp**, чье значение всегда равно TRUE (поскольку сама панель носит поясняющий характер, эти лампы будут гореть всегда); к серой лампе переменных привязано не будет.

Переменную можно привязать сразу к группе выделенных графических элементов:

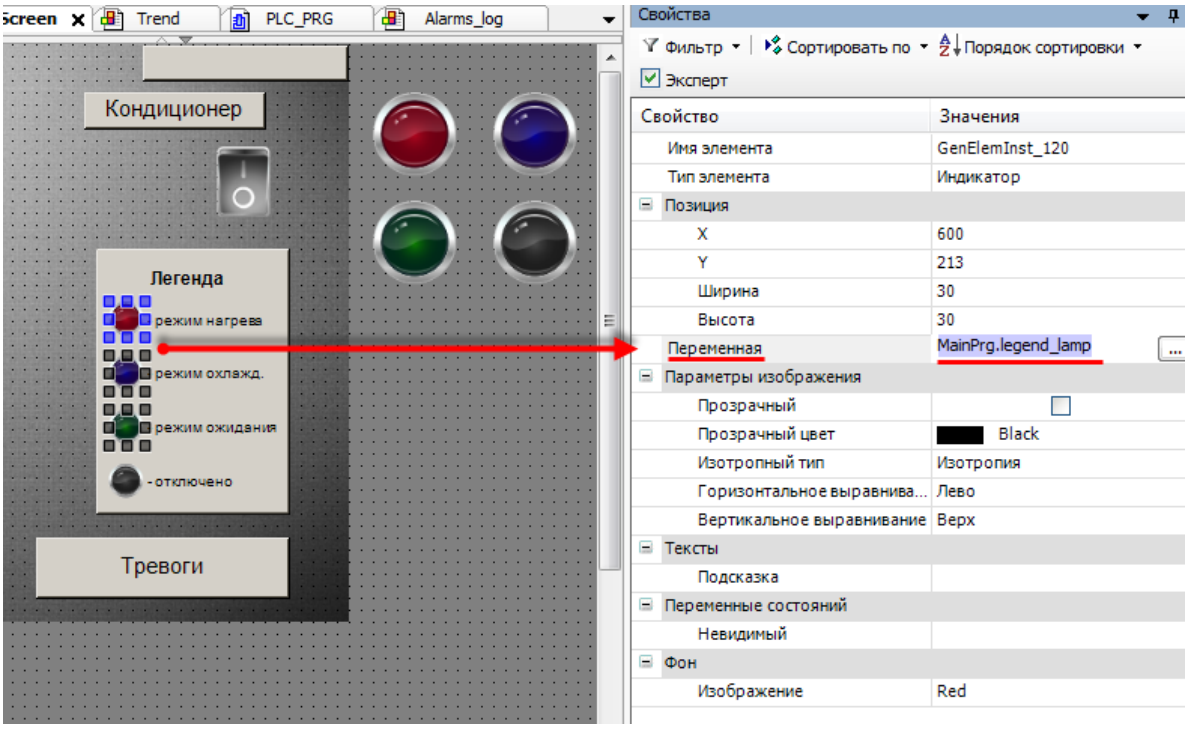


Рис. 7.112. Привязка переменной к группе элементов

Итак, на данный момент на экране **MainScreen** привязаны следующие переменные:

Табл. 4. Переменные, привязанные к графическим элементам экрана **MainScreen**

(после п. 7.5.1., пп. I)

Элемент	Параметр	Привязанная переменная
Отображение линейки	Значение	temp
Чекбокс (кнопка-флажок)	Переменная	measure_mode
Клавишный выключатель	Переменная	conditioner_power
Большие индикаторы		
Красный индикатор (red)	Переменная	red_light
	Переменная состояния (невидимость)	red_inv
Синий индикатор (blue)	Переменная	blue_light
	Переменная состояния (невидимость)	blue_inv
Зеленый индикатор (green)	Переменная	green_light
	Переменная состояния (невидимость)	green_inv
Серый индикатор (gray)	Переменная состояния (невидимость)	gray_inv
Малые индикаторы (панель Легенда)		
Красный индикатор (red)	Переменная	legend_lamp
Синий индикатор (blue)	Переменная	legend_lamp
Зеленый индикатор (green)	Переменная	legend_lamp
Серый индикатор (gray)	-	-

II. Вывод значений переменных

На панели **Режим управления** размещены три **поля вывода** значений параметров: температуры, ее уставки и уставки гистерезиса. Помимо привязки к ним переменных, необходимо также настроить **формат вывода** значений, который указывается во вкладке **Тексты**.

Настроим вывод значения температуры следующим образом:

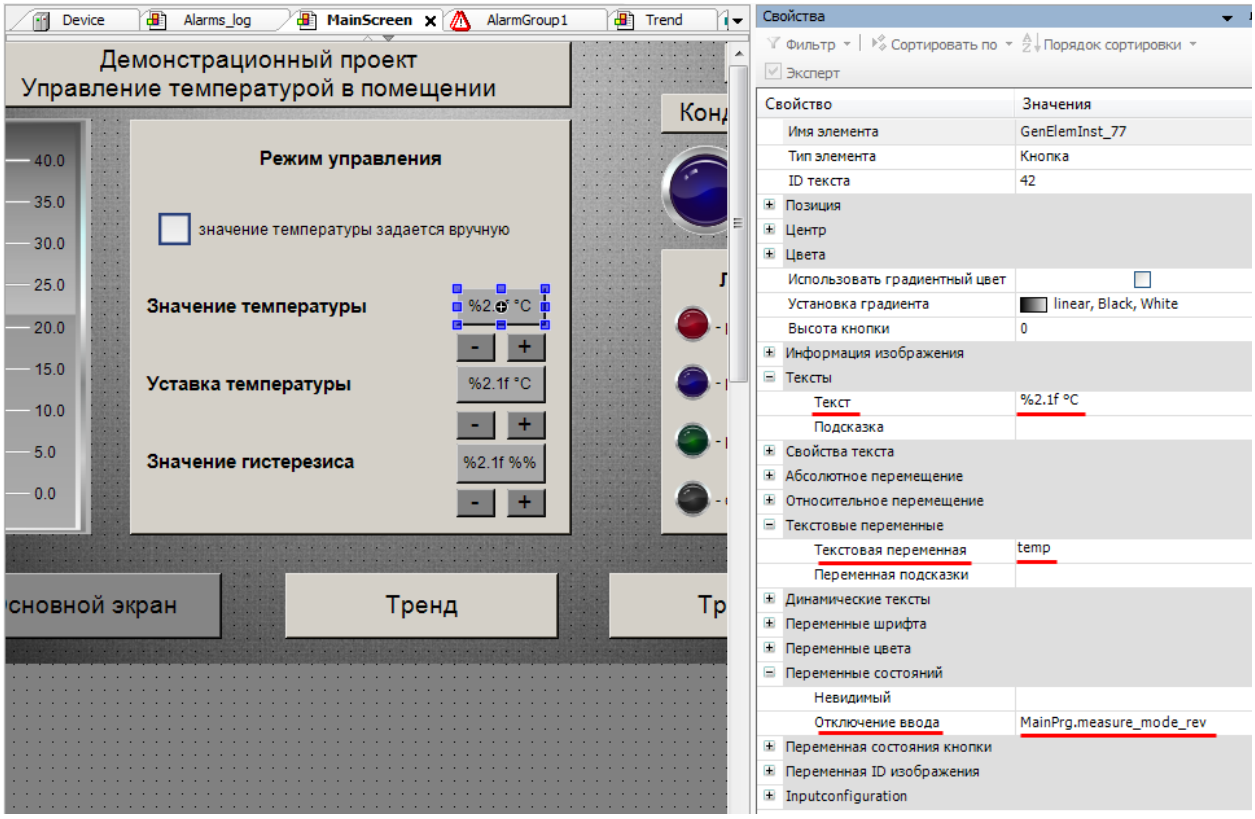


Рис. 7.113. Настройки вывода значения переменной temp

В поле **Отключение ввода** вкладки **Переменные состояний** указывается переменная, которая, принимая значение TRUE, делает данный элемент неактивным для управления. Поскольку в режиме эмуляции значение температуры будет задаваться пользователем с помощью нажатия на данное поле вывода (или на расположенные под ним кнопки), то для того, чтобы исключить эту возможность при получении значения температуры с датчика, привяжем к данному параметру переменную **Measure_mode_rev**.

В поле **Текстовая переменная** указывается переменная, чье значение будет выводиться данным элементом. В данном случае это переменная **temp**, характеризующая текущее значение температуры.

В поле **Текст** указывается **форматирование** вывода. Общий синтаксис форматирования следующий:

<Текст> %<мин. ширина>.<точность><спецификатор переменной> <Текст>

где

<Текст> - дополнительный текст (например, название и размерность переменной);

% - спецсимвол, после которого указывается тип переменной (перед типом могут быть указаны настройки вывода). Если необходимо вывести символ процента в качестве текста, его следует записать как %%.

<мин. ширина> - ширина поля вывода;

<точность> - количество выводимых символов после запятой (для целочисленных переменных), количество выводимых символов (для строковых переменных);

<спецификатор переменной> - определяет тип выводимой переменной.

Для значения температуры мы используем форматирование **%2.1f °C**, т.е. тип данных – действительное число, с отображением одного знака после запятой.

Для корректного отображения значений, **спецификатор** переменной должен соответствовать **типу данных**, которому принадлежит переменная (см. табл. 1). Список спецификаторов приведен в табл. 5:

Табл. 5. Спецификаторы типов переменных

Символ	**Тип отображаемых данных**	**Пример использования**		
		Фактическое значение	**Форматирование**	**Отображаемое значение**
%d	десятичное число	10	%d	10
%b	двоичное число	10	%b	00000000 00001010
%o	восьмеричное число без знака (без ведущего нуля)	10	%o	12
%x	шестнадцатеричное число без знака (без ведущего нуля)	10	%x	a
%u	десятичное число без знака	-10	%u	10
%c	символ таблицы ASCII (переменная типа byte)	33	%c	!
%s	строка	abcdef	%s %.3s	abcdef abc
%f	действительное число	10.1111	%f %2.2f	10.111100 10.11
%t	системное время	12:48:52	%t[hh:mm:ss]	12:48:52

Поля вывода значений уставки температуры и гистерезиса настроим ***по аналогии*** с выводом температуры – привяжем к ним текстовые переменные **temp_ust** и **hyst**. Поле **Отключение ввода** остается незаполненным, т.к. управление этими переменными не зависит от режима измерения.

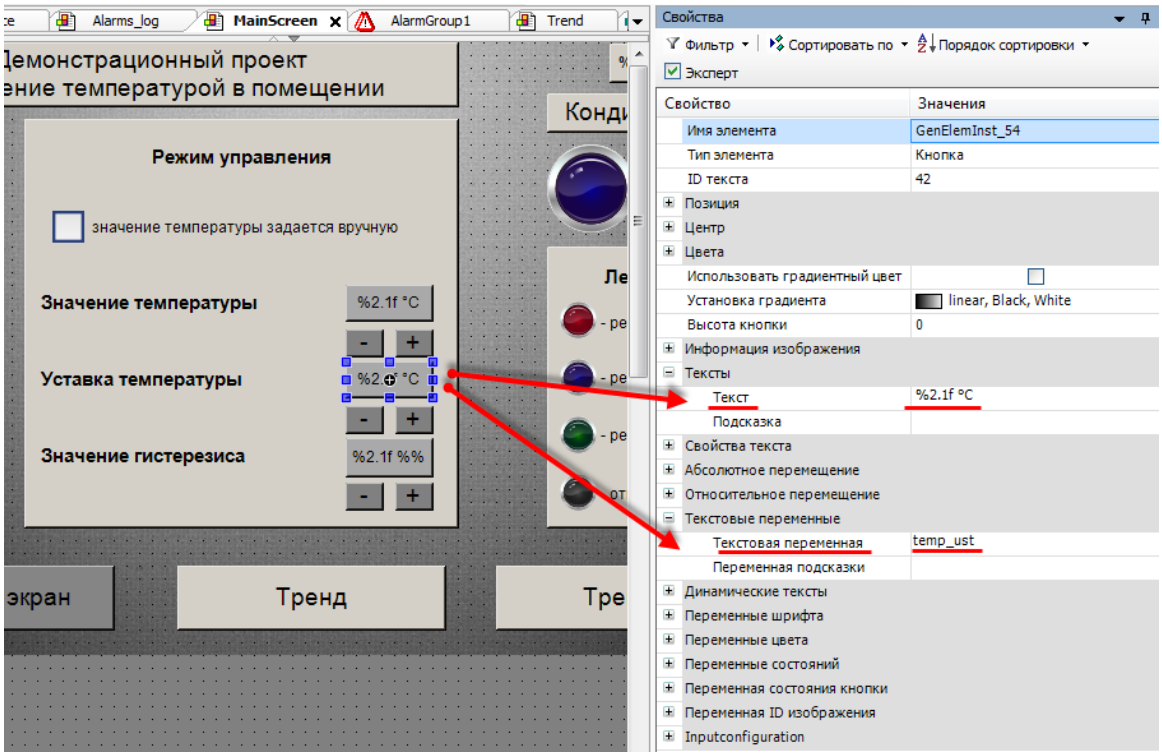


Рис. 7.114. Настройки вывода значения переменной **temp_ust**

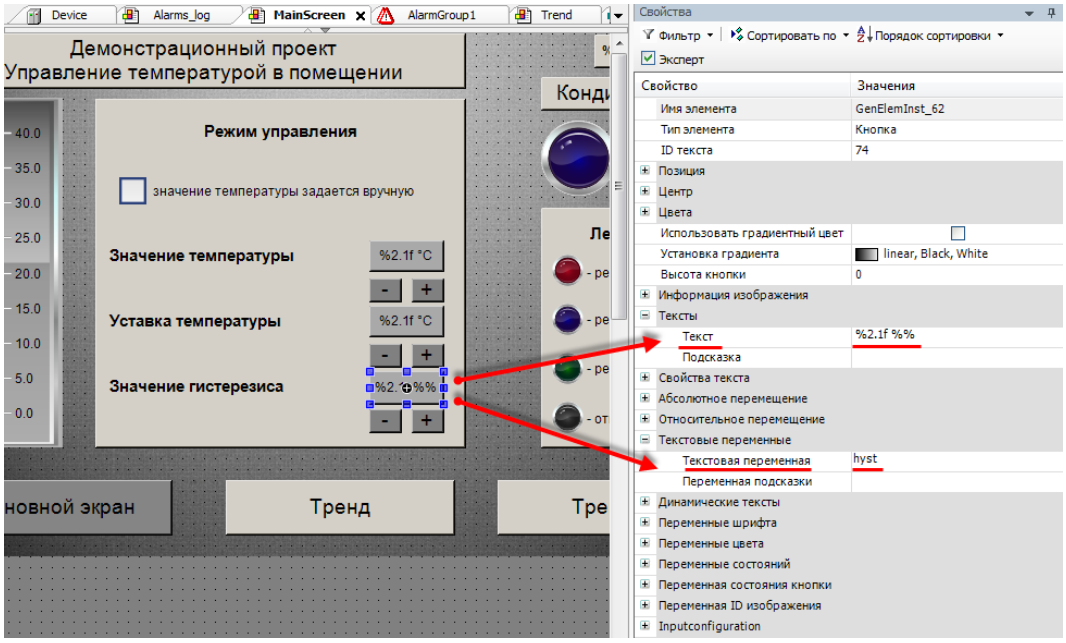


Рис. 7.115. Настройки вывода значения переменной **hyst**

Помимо **вывода** значений параметров, эти поля будут использоваться и для их **ввода**; необходимые для этого действия будут описаны в [п. III](#).

Последняя переменная, значение которой будет отображаться на экране **MainScreen** – это системное время. Для вывода времени существует внутренняя переменная, для обращения к которой достаточно использовать соответствующее форматирование: %t[…].

В квадратных скобках указываются нужные **заполнители**, например, в нашем случае это

%t[dd.MM.yyyy HH:mm:ss]

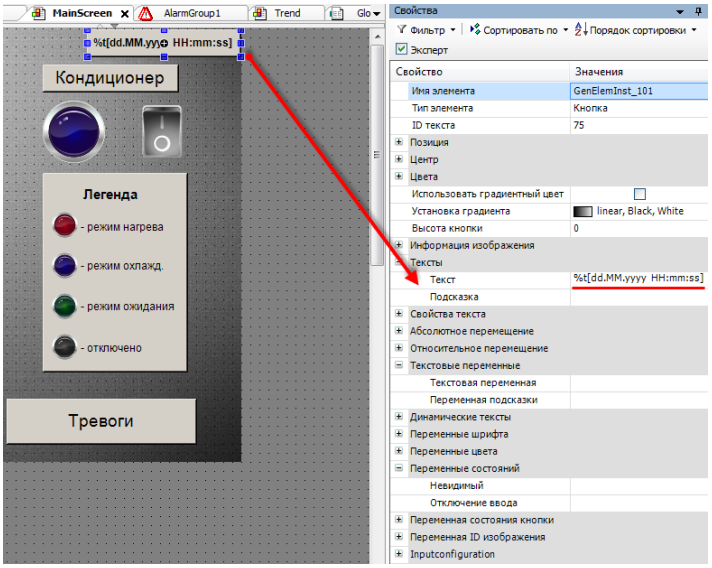


Рис. 7.116. Настройки вывода системного времени

Полный список заполнителей приведен в табл. 6:

Табл. 6. Заполнители форматирования времени

Заполнитель	**Отображаемое значение**	**Пример отображения**
ddd	Сокращенное название дня недели	Fri (пятница)
dddd	Полное имя дня недели	Monday (понедельник)
ddddd	День недели в виде числа	0 (воскресение), 1 (понедельник)
MMM	Сокращенное название месяца	Feb (февраль)
MMMM	Полное название месяца	February (февраль)
d	Число месяца в виде числа (1 – 31)	8
dd	Число месяца десятичным числом (01 – 31),	08
M	Месяц в виде числа (1 – 12)	8
MM	Месяц в виде десятичного числа (01 – 12),	08
jjj	День в году в виде десятичного числа (001-366)	253
y	Год века (0 – 99), например, "9"	8
yy	Год века десятичным числом (00 – 99)	08
yyy	Полный год	2008
HH	Час в 24-часовом формате (01– 24),	8
hh	Час в 12-часовом формате (01 – 12),	8 (и для 8-00, и для 20-00)
m	Минуты (0 – 59)	8
mm	Минуты десятичным числом (00 – 59),	08
s	Секунды (0 – 59)	8
ss	Секунды десятичным числом (00 – 59),	08
ms	Миллисекунды (0 – 999)	888
t	Идентификатор для 12-часового формата: A (часы <12) и P (часы >12)	A (8 часов)
tt	Идентификатор для 12-часового формата: AM (часы <12) и PM (часы >12)	PM (15 часов)

Если помимо времени необходимо выводить сопроводительный текст, который содержит заполнители (например, Las**t** Up**d**a**t**e), то следует заключать этот текст в одиночные кавычки (‘Last Update’).

Настроим вывод системного времени на всех трех экранах проекта (см. рис. 7.116).

III. Настройка действий

На экране **MainScreen** расположено 11 кнопок, для которых будут настроены действия: 2 кнопки перехода, 3 кнопки ввода значений, 6 кнопок «увеличить»/ «уменьшить»:

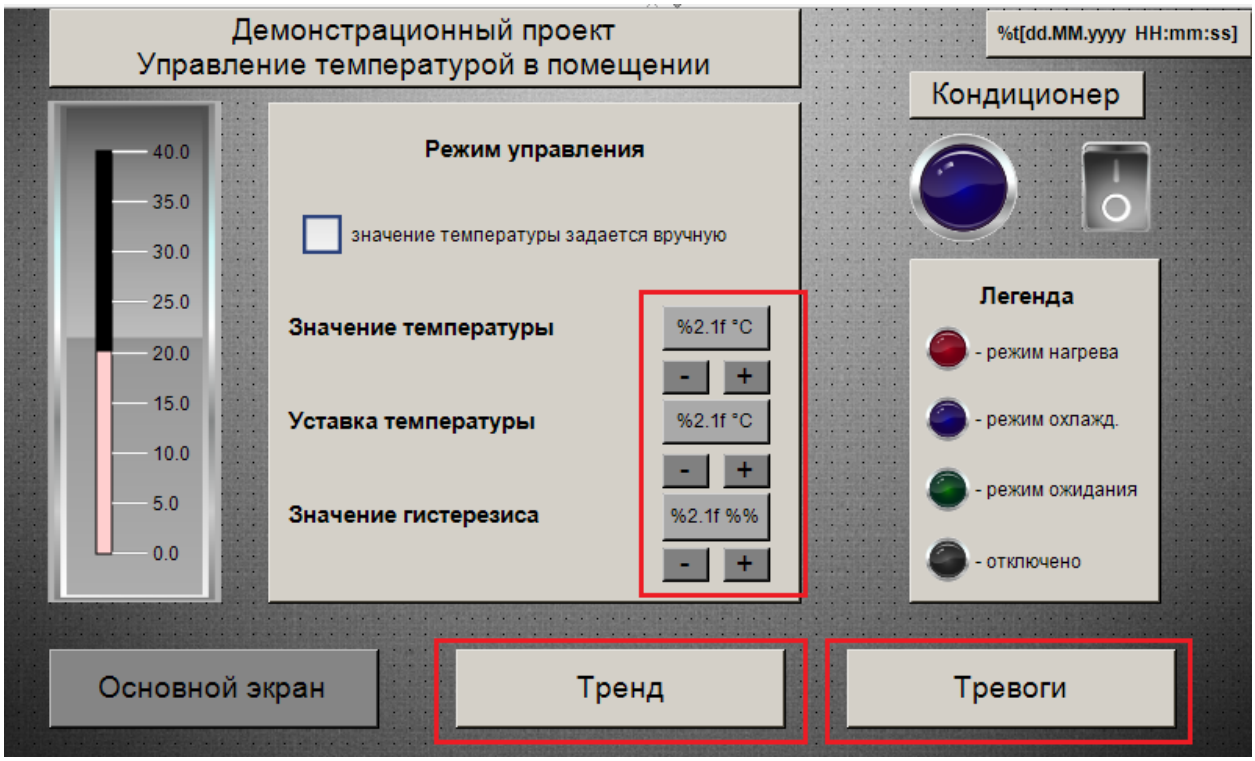


Рис. 7.117. Кнопки экрана **MainScreen**, для которых будут настроены действия

Для начала настроим кнопки перехода с экрана **MainScreen** на экраны **Trend** и **Alarms_log**. Переход будет осуществляться ***при нажатии ЛКМ*** на соответствующую кнопку. ***Очевидно***, что настраивать переход с экрана **MainScreen** на экран **MainScreen** не нужно.

Выделим кнопку **Тренд,** выберем в ее настройках вкладку **Inputconfiguration** и нажмем **ЛКМ** на поле **OnMouseClick**. Откроется диалоговое окно **Конфигурации ввода**.

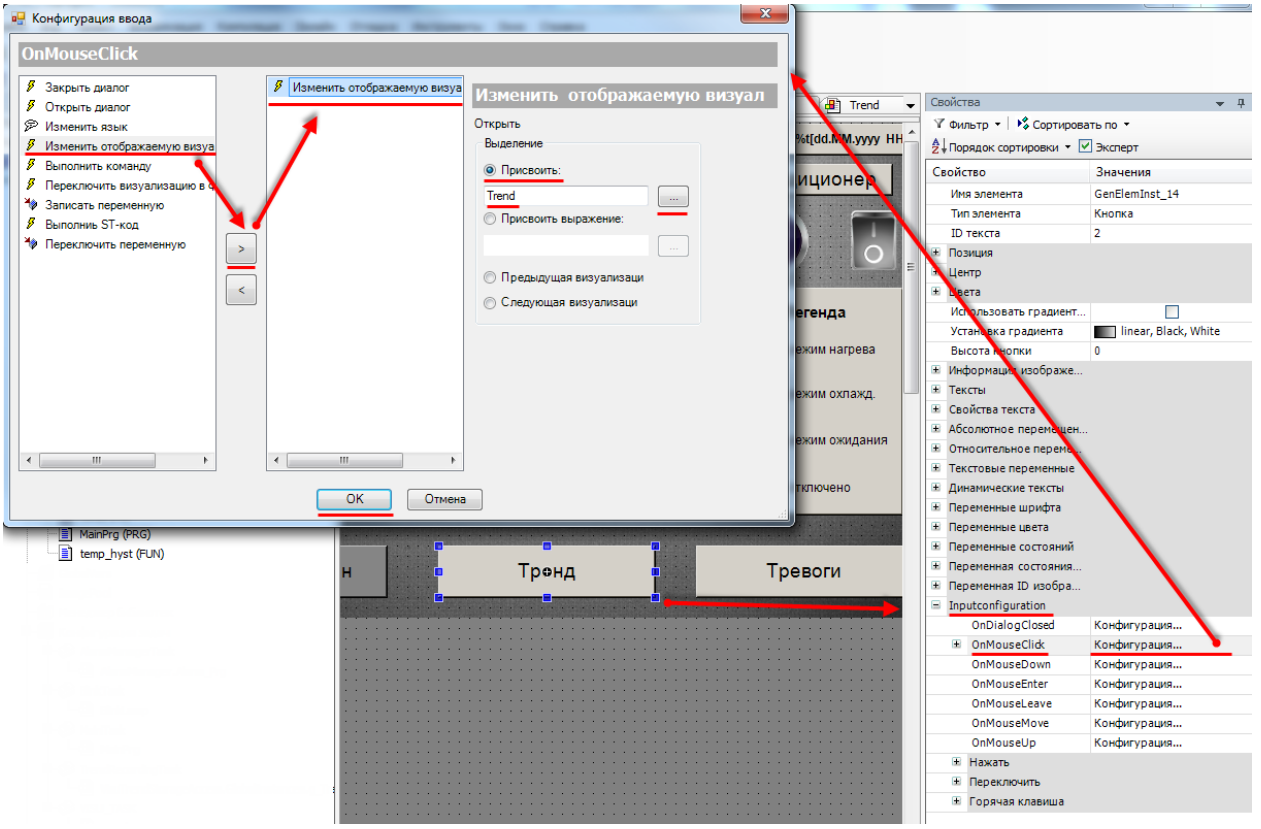


Рис. 7.118. Диалоговое окно **Конфигурации ввода** (изменение отображаемой визуализации)

Выберем на левой панели действие **Изменить отображаемую визуализацию**, с помощью кнопки «**>**» присвоим его кнопке **Тренд**, и в настройках действия выберем экран, на который должен осуществлять переход – в нашем случае, это экран **Trend**. Нажмем **ОК**.

Аналогичным образом настроим кнопку **Тревоги** (переход на экран **Alarms_log**). Настроим кнопки перехода на остальных экранах проекта.

Теперь настроим поля вывода параметров таким образом, чтобы по нажатию **ЛКМ** на них открывалось диалоговое окно **задания значения переменной**.

Выделим поле вывода **Значение температуры**, выберем в его настройках вкладку **Inputconfiguration** и нажмем **ЛКМ** на поле **OnMouseClick**. Откроется диалоговое окно **Конфигурации ввода**.

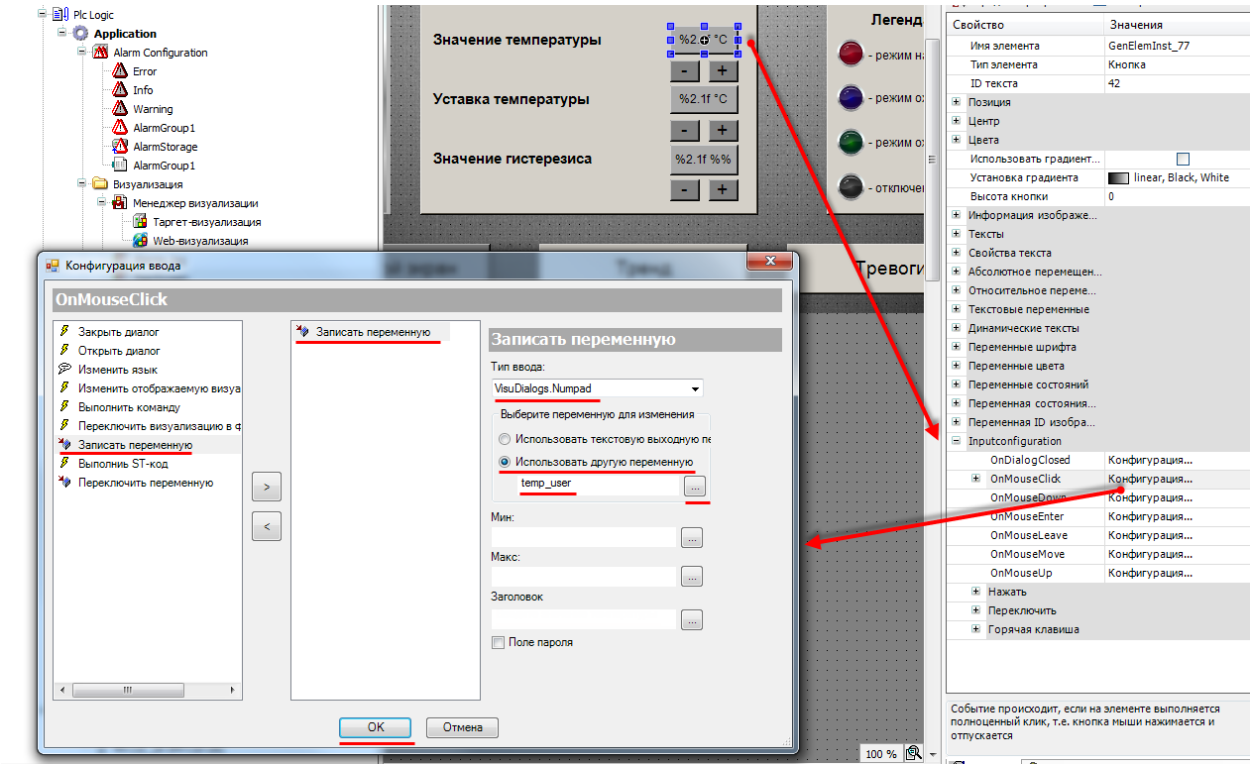


Рис. 7.119. Диалоговое окно Конфигурации ввода (запись переменной)

Выберем на левой панели действие **Записать переменную**, с помощью кнопки «**>**» присвоим его выделенному полю ввода, и в настройках действия выберем **тип клавиатуры** (в нашем случае – цифровая) и записываемую переменную (поскольку задание значения температуры возможно только в режиме эмуляции, мы используем переменную **temp_user**, а не temp). Нажмем **ОК**.

Аналогично настроим поля ввода **Уставка температуры** (запись переменной **temp_ust**) и **Значение гистерезиса** (запись переменной **hyst**); для гистерезиса также зададим нижний и верхний предел записи переменной – 0 и 100 соответственно (т.к. гистерезис задается в процентах относительно температуры уставки).

Осталось настроить кнопки «уменьшить»/«увеличить» ().

Выделим кнопку **«уменьшить»,** расположенную под полем вывода значения температуры, выберем в ее настройках вкладку **Inputconfiguration** и нажмем **ЛКМ** на поле **OnMouseClick**. Откроется диалоговое окно **Конфигурации ввода**.

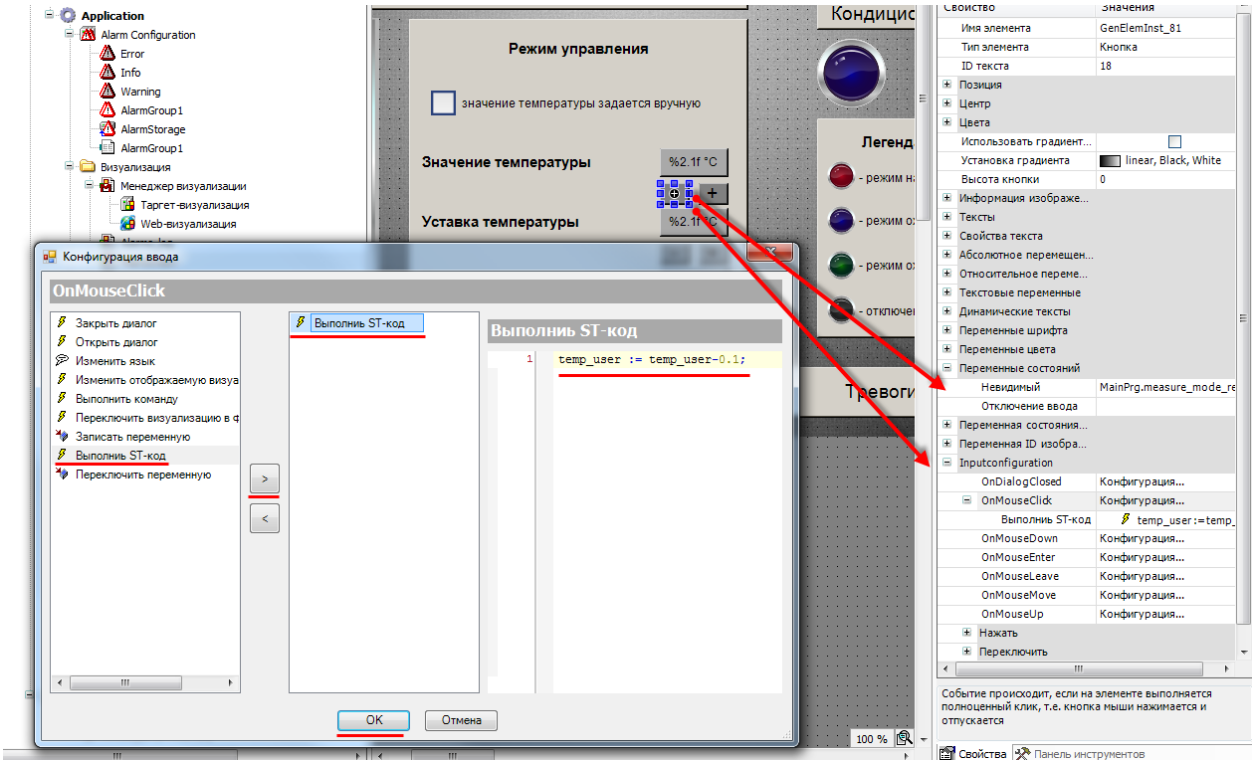


Рис. 7.120. Диалоговое окно **Конфигурации ввода** (выполнение ST-кода)

Выберем на левой панели действие **Выполнить ST-код**, с помощью кнопки «**>**» присвоим его выделенному полю ввода и напишем **код на языке ST,** который будет выполняться при нажатии кнопки. Очевидно, в нашем случае это

temp_user := temp_user-0.1;

т.е. каждое нажатие на кнопку «уменьшить» будет приводить к уменьшению температуры на 0.1 градус Цельсия.

Так как задавать значение температуры можно только в режиме эмуляции,то в настройках кнопки во вкладке **Переменные состояний** привяжем к параметру **Невидимый** переменную **measure_mode_rev**; это приведет к тому, что в режиме «измерение с датчика», кнопки «увеличить»/«уменьшить» отображаться не будут. Невидимость носит **функциональный**, а не визуальный характер – т.е. нажатие на место расположения невидимой кнопки ***не приведет*** к выполнению какого-либо действия.

Аналогичным образом настроим **кнопку «увеличить».** Очевидно, единственным отличием будет знак в программном коде:

```
temp_user := temp_user+0.1;
```

Настроим кнопки «увеличить»/«уменьшить» для уставки температуры и гистерезиса. Для этих кнопок настраивать **невидимость** уже **не нужно**. Исполняемый ими код будет выглядеть следующим образом:

для температуры уставки:

```
temp_ust := temp_ust-0.1;
```

```
temp_ust := temp_ust+0.1;
```

для значения гистерезиса:

```
hyst:=hyst-0.1;
```

```
hyst:=hyst+0.1;
```

Вкладка **Inputconfiguration** предоставляет возможность настройки различных вариантов по взаимодействию с кнопками - существует возможность привязывать действия не только к их нажатию, но и зажатию, отпусканию, наведению курсора мыши и т.д. Список возможных действий также не ограничивается упомянутыми выше. В рамках учебного проекта они рассматриваться ***не будут***; подробную информацию и о них можно найти в **справке CODESYS**.

7.5.2. Экран Trend

I. Привязка переменных, настройка кнопок

Сначала повторим те действия, которые нам уже знакомы:

1. Настроим кнопки перехода на другие экраны (см. п. 7.5.1 пп. III) и отображение системного времени (см. п. 7.5.1 пп. II);

2. Привяжем к желтой аварийной лампе переменную yellow_lamp (см. п. 7.5.1 пп. I);

3. Настроим поле для **ввода и отображения** названия лампы следующим образом (см. п. 7.5.1 пп. III):

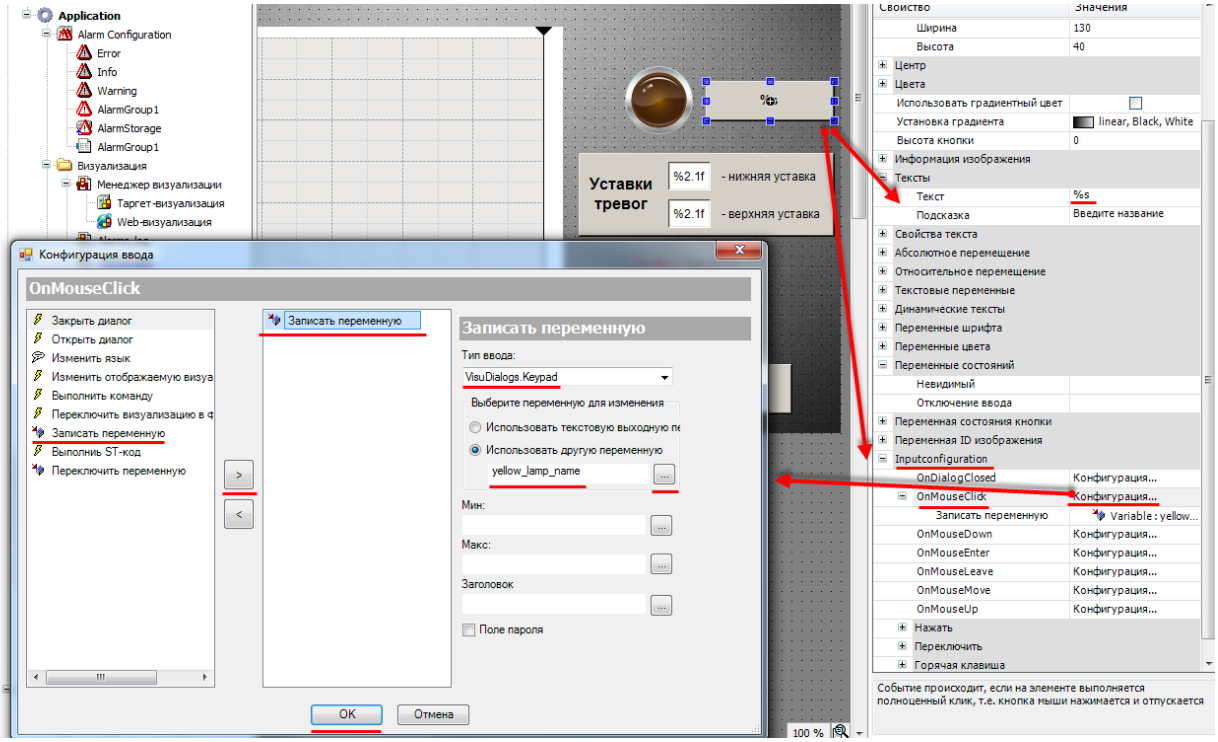


Рис. 7.121. Настройка поля ввода/отображения названия сигнальной лампы

4. Настроим поля ввода и отображения значений уставок тревог. Форматирование вывода - %2.1f, конфигурация действия следующая:

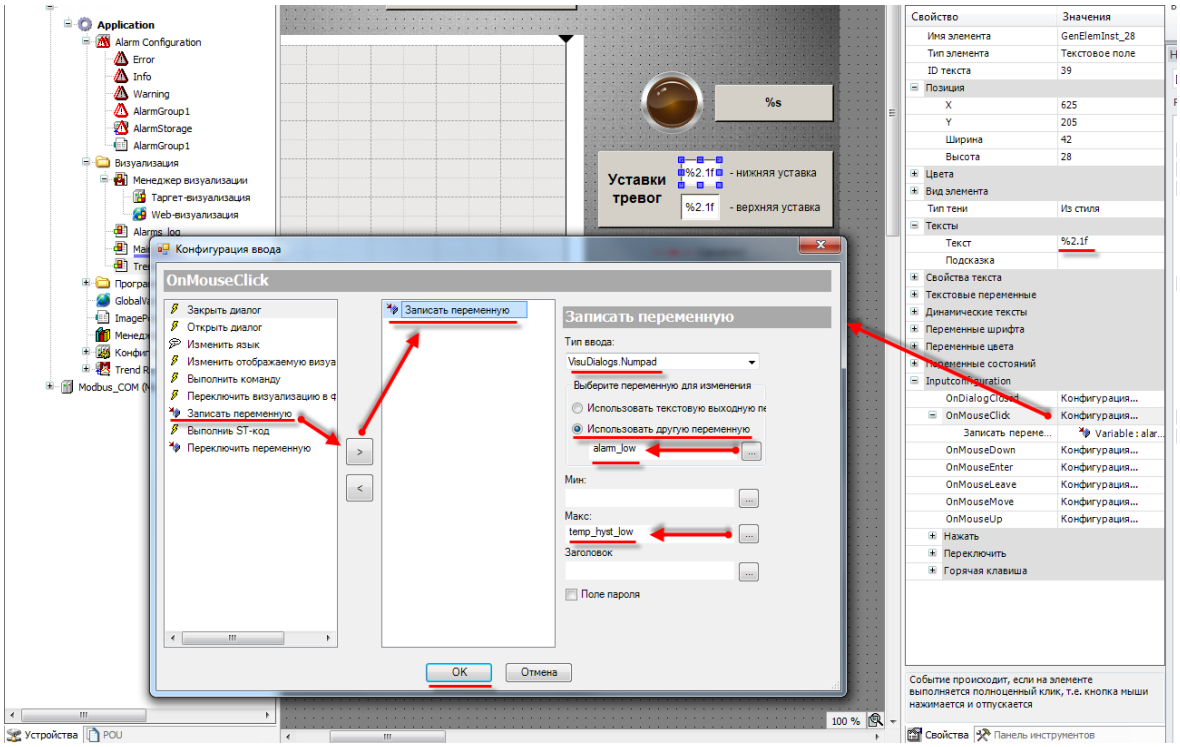


Рис. 7.122. Настройка действия поля **нижней** уставки тревоги

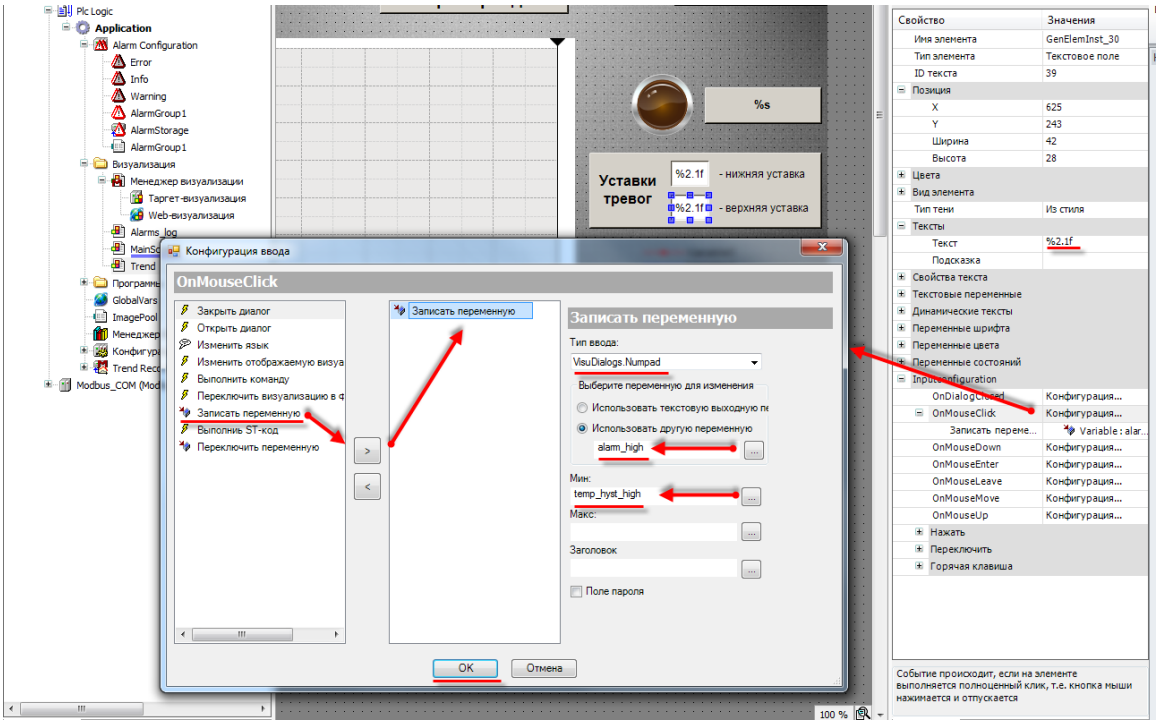


Рис. 7.123. Настройка действия поля **верхней** уставки тревоги

II. Настройка элемента Тренд

Для настройки тренда нажнем на него **ПКМ** и в контекстном меню выберем пункт **Edit trend recording** (или нажмем на компонент **Trend_Trend1** на **Панели устройств**):

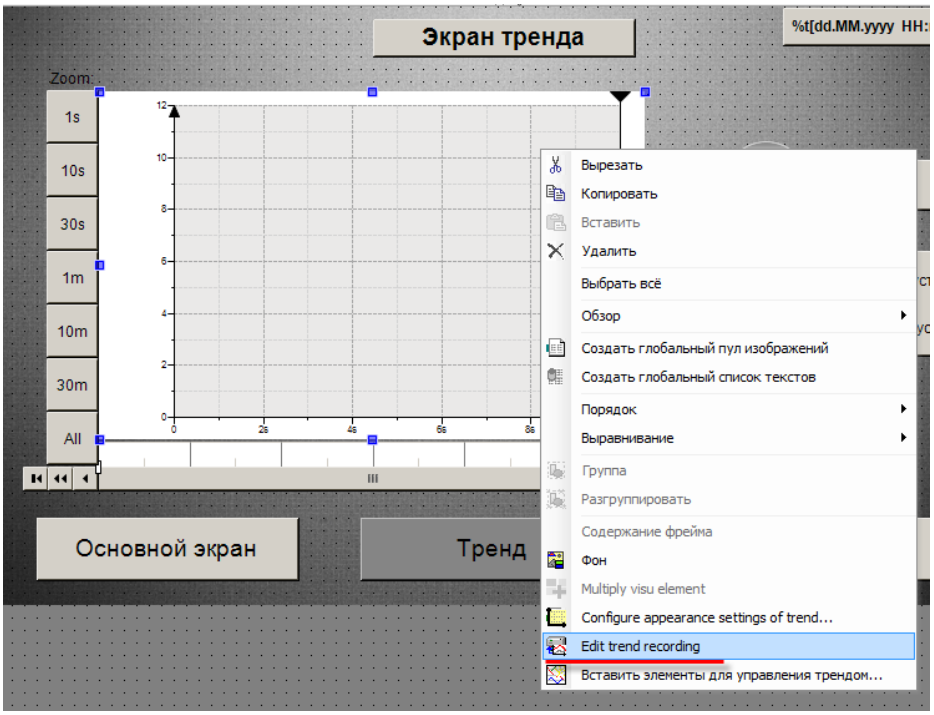


Рис. 7.124. Открытие окна конфигурации тренда

Появится **окно конфигурации тренда**, которое выглядит следующим образом:

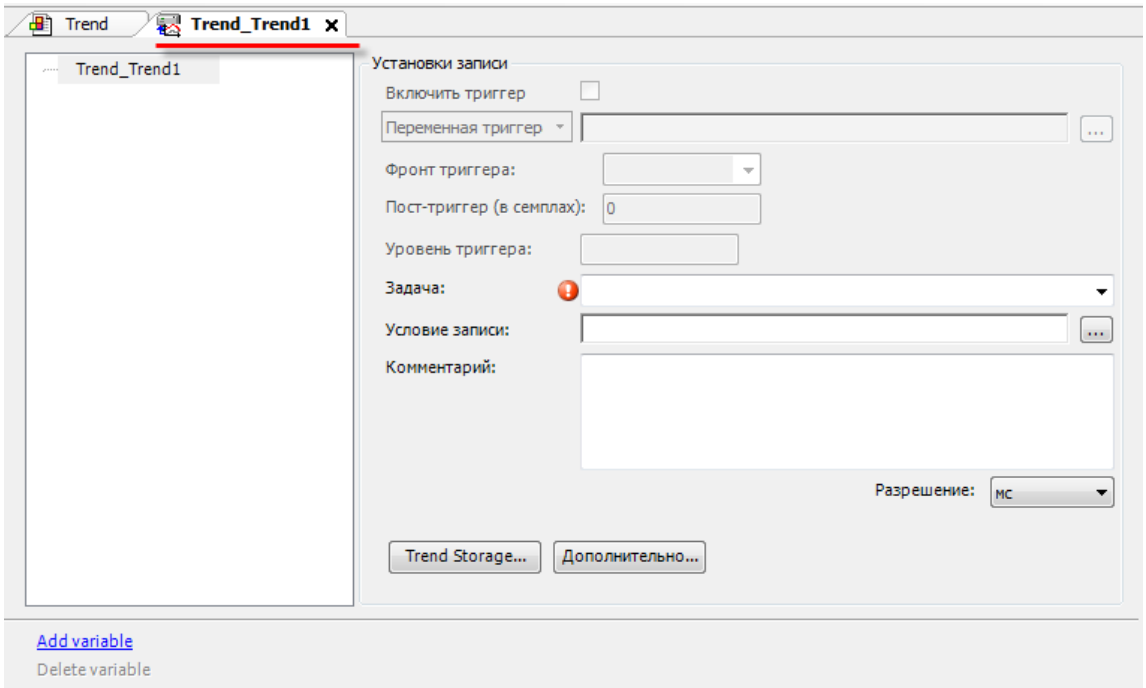


Рис. 7.125. Окно конфигурации тренда

В поле **Задача** необходимо указать задачу, к которой будет привязан тренд. Подробнее настройка задач будет описана в п. 7.7, пока же просто выберем задачу, которая автоматически добавилась в проект после создания тренда – **TrendRecordingTask**.

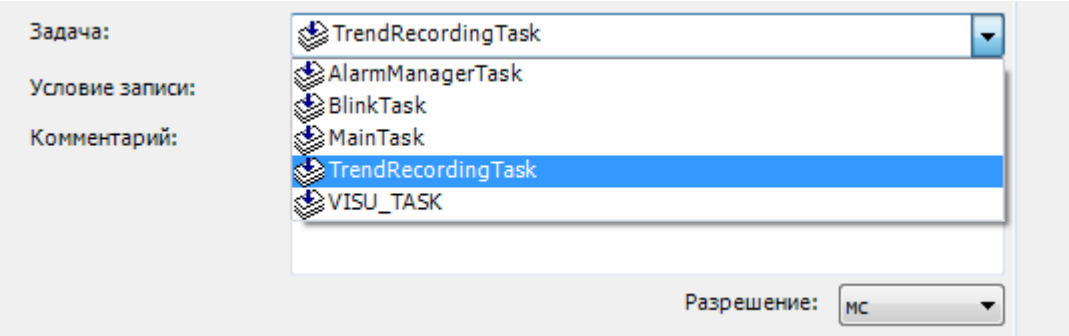


Рис. 7.126. Выбор задачи тренда

Вкладка **Trend Storage** позволяет настроить **параметры архивации** тренда - такие, как число записываемых переменных, размер файла записи и т.д.

Теперь добавим переменные, которые будут отображаться на тренде. Нажмем кнопку **Add Variable** и выберем переменную **Temp**.

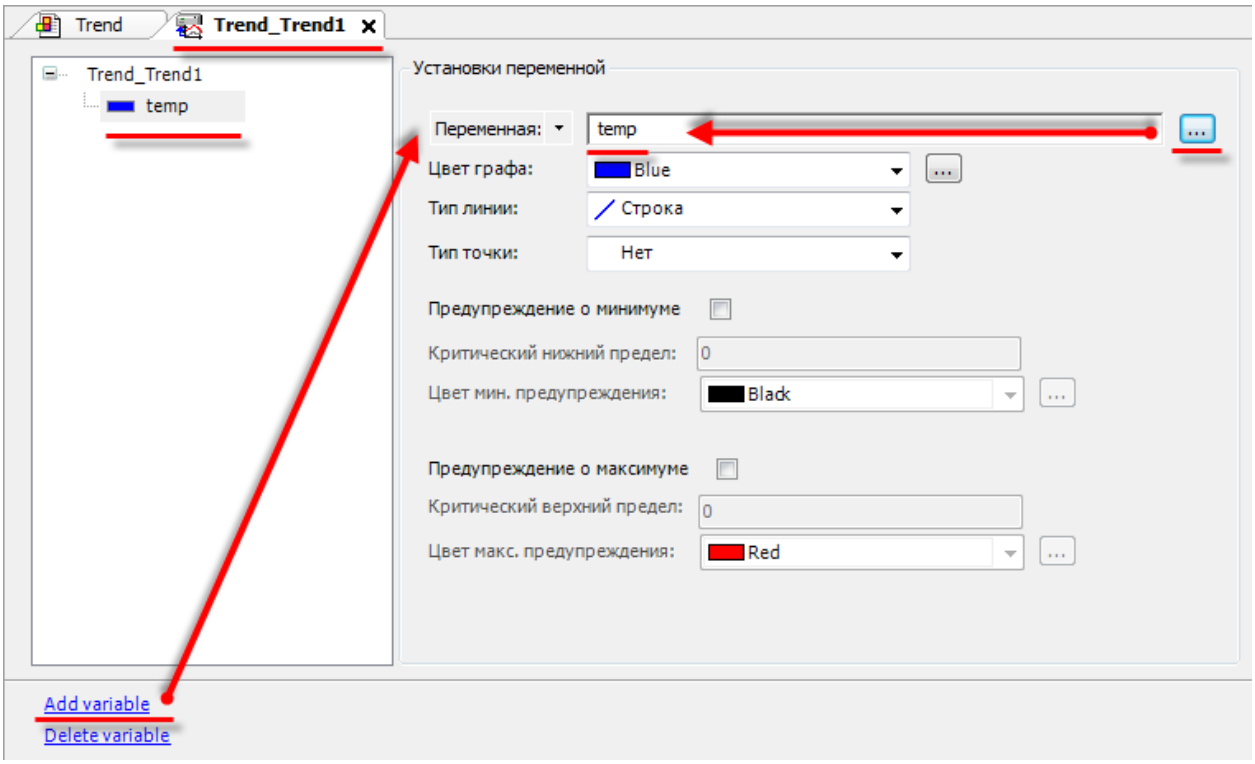


Рис. 7.127. Добавление переменной на тренд

Аналогичным образом добавим переменные **temp_ust**, **temp_hyst_low**, **temp_hyst_high**, **alarm_low, alarm_high**. Зададим им различные цвета, границам гистерезиса укажем **Тип точки** cross.

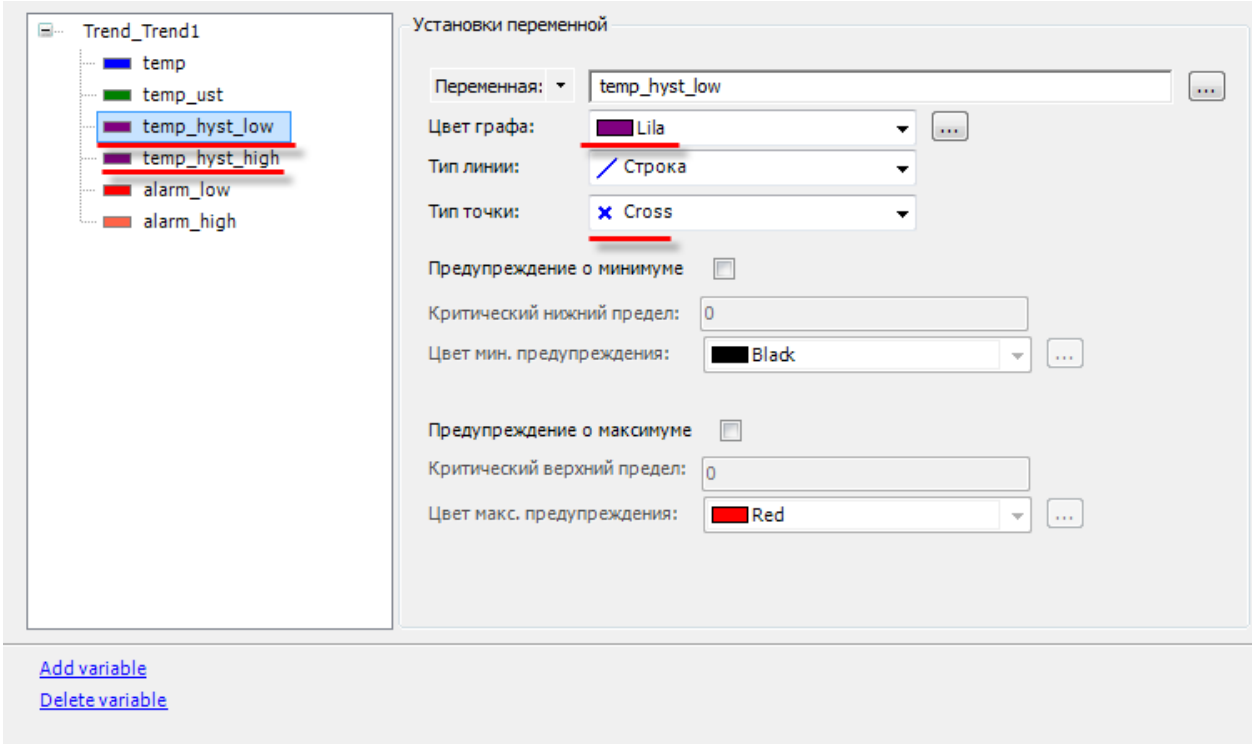


Рис. 7.128. Сконфигурированный тренд

Так как автоматически созданные дополнительные элементы (легенда, селектор времени, селектор даты; см. п. 7.3.3) настройки не требуют, то на этом конфигурирование тренда завершено.

7.5.3. Экран Alarms_log

Таблица тревог не требует привязки переменных; для ее работы необходимо добавить и настроить компонент **Конфигуратор тревог**. Этот процесс будет описан в следующем пункте (см. п. 7.6). Автоматически созданные кнопки (Подтвердить, История и т.д.) настройки не требуют.

Соответственно, единственное, что нужно сделать на этом экране – настроить отображение системного времени (см. рис. 7.116).

7.6. Настройка конфигуратора тревог

7.6.1. Добавление Конфигуратора тревог в проект

При необходимости ведения в проекте **Журнала событий** (частным случаем которого является **Журнал тревог**) используется компонент **Конфигуратор тревог**, который обеспечивает работу графических примитивов **Таблица тревог** и **Баннер тревог**.

В нашем проекте мы будем использовать тревоги для отображения информационных сообщений в **Таблице тревог** в случае выхода значения температуры за **границы гистерезиса** и **аварийные уставки**.

Добавим в проект компонент **Конфигуратор тревог** (его добавление приведет также к автоматическому созданию соответствующей **задачи**):

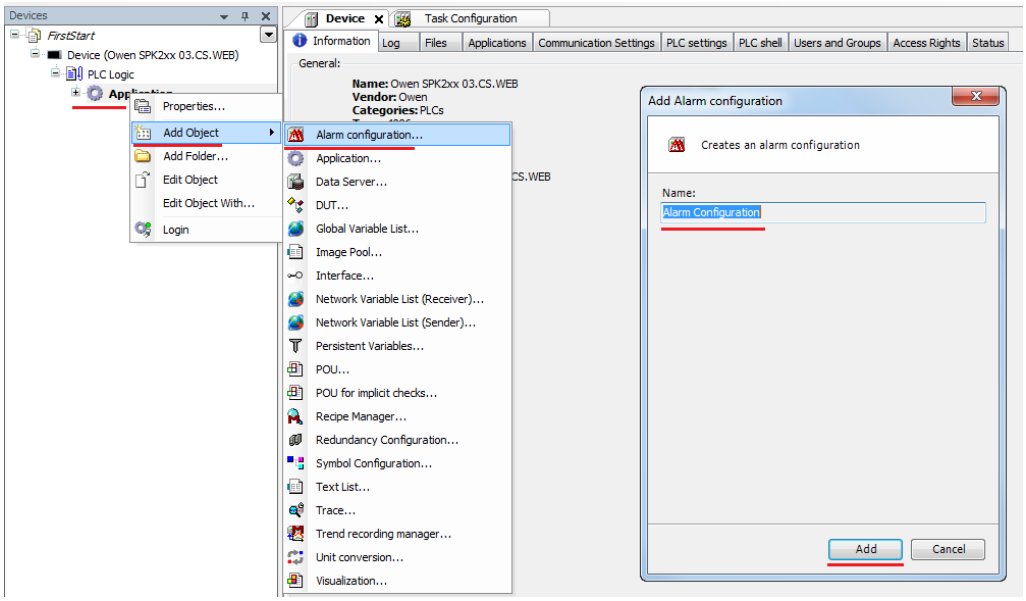


Рис. 7.129. Добавление в проект **Конфигуратора тревог**

//// В текущей версии CODESYS имеются проблемы с добавлением Конфигуратора тревог при использовании русскоязычного интерфейса среды программирования. В этом случае рекомендуется переключить язык интерфейса на английский (см. рис. 3.8), добавить Конфигуратор тревог и переключить язык интерфейса на русский. ////

По умолчанию **Конфигуратор тревог** содержит **четыре дочерних компонента**: **три класса тревог** (Error, Info, Warning) и **хранилище тревог** Alarm Storage. Добавим еще один дочерний компонент – **группу тревог** с названием **AlarmGroup1**. При ее добавлении ***автоматически*** создастся **список текстов** с идентичным названием.

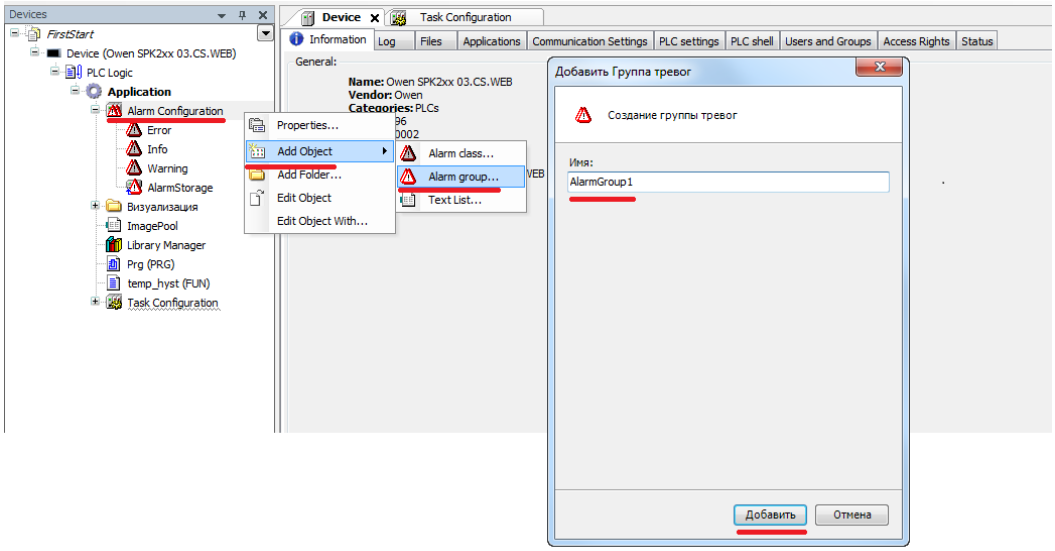


Рис. 7.130. Добавление группы тревог **AlarmGroup1**

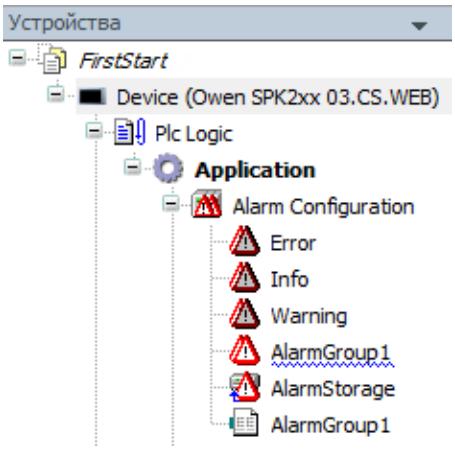


Рис. 7.131. Внешний вид **Конфигуратора тревог** на **Панели устройств**

Рассмотрим дочерние компоненты **Конфигуратора тревог**:

1. **Классы тревог** предназначены для разделения тревог на отдельные типы. Иными словами, они определяют набор базовых параметров (таких, как приоритет, способ подтверждения и т.д.), который может соответствовать как одной, так и нескольким тревогам. Пользователь может создавать собственные классы тревог, помимо имеющихся по умолчанию;

2. **Группы тревог** представляют собой наборы тревог, которые могут относиться как к одному, как и к разным классам. На уровне группы настраиваются индивидуальные параметры тревоги - условия появления и пропадания, текст сообщения и т.д.;

3. **Список текстов** содержит тексты сообщений тревог и их идентификаторы;

4. **Хранилище тревог** обеспечивает их запись в базу данных.

7.6.2. Настройка классов тревог

Рассмотрим настройки **класса тревог** на примере созданного по умолчанию класса **Error:**

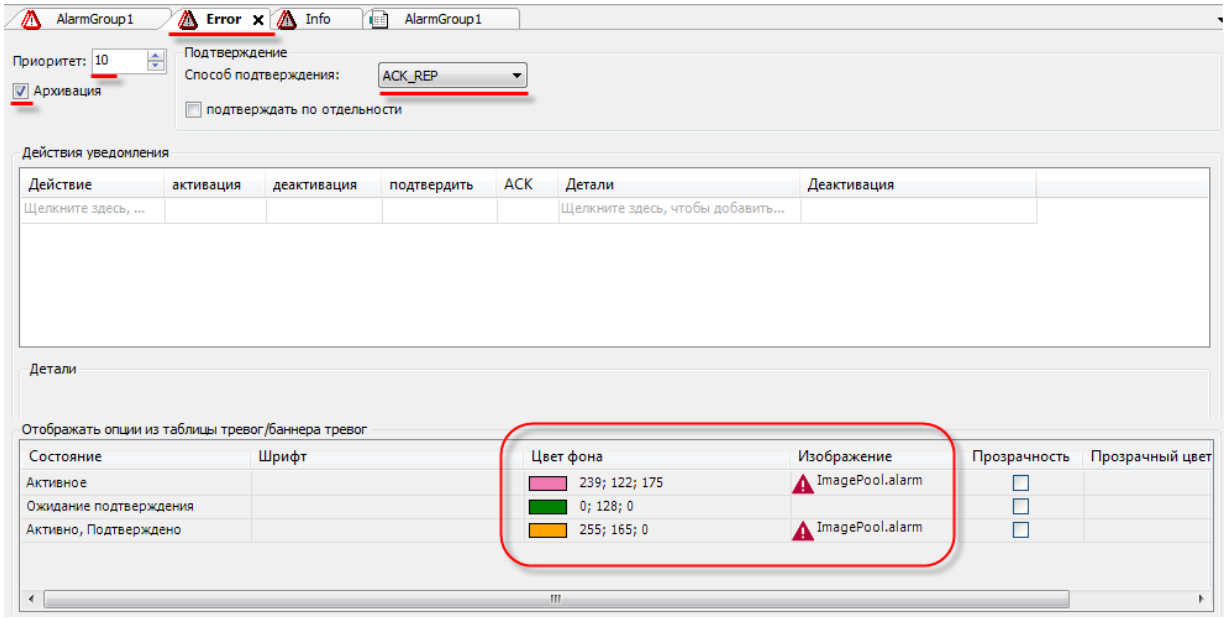


Рис. 7.132. Настройки класса тревог Error

1. **Приоритет** характеризует степень важности тревоги. Это информационный параметр, который может быть отображен в **Таблице тревог**. Самый высокий приоритет – 0, самый низкий – 255;

2. **Способ подтверждения** влияет на условие пропадания тревоги из **Таблицы тревог**. Подтверждение в общем случае подразумевает нажатие пользователем на кнопку **Подтвердить** или **Подтвердить всё**;

Табл. 7. Способы подтверждения тревог

Способ подтверждения тревоги	**Возможные состояния тревоги**	**Условие исчезновения тревожного сообщения**
REP	Активное	Условие появления тревоги перестало выполняться
ACK	Активное	Тревога подтверждена пользователем
REP_ACK	Активное, Ожидание подтверждения	Условие появления тревоги перестало выполняться***, после чего*** она была подтверждена пользователем
ACK_REP	Активное, Ожидание Подтверждения, Подтвержденное	Условие появления тревоги перестало выполняться, и она была подтверждена пользователем (порядок этих событий не имеет значения)
ACK_REP_ACK	Активное, Ожидание Подтверждения, Подтвержденное	Условие появления тревоги перестало выполняться, после чего она была подтверждена пользователем **или** активная тревога была подтверждена пользователем, после чего условие ее появления перестало выполняться, и это было также подтверждено пользователем

При наличии галочки **Подтверждать по отдельности** у пользователя пропадает возможность подтвердить сразу все тревоги;

3. Чекбокс **Архивация** отвечает за сохранение тревог в памяти контроллера;

4. Меню **Действие уведомления** позволяет задать список действий, которые выполняются при смене состояний тревоги. Количество состояний тревоги зависит от способа ее подтверждения;

5. Меню **Отображать опции** позволяет настроить отображение тревог в **Таблице/Баннере тревог**, в частности, задать ее состояниям индивидуальный шрифт, цвет фона, пиктограмму и т.д. Для задания пиктограммы нужно кликнуть на соответствующее поле, ввести имя пиктограммы (любое), после чего нажать на enter и указать путь к соответствующему изображению (поддерживаются все распространенные графические формы типа .jpg, .png, .bmp и т.д.). При необходимости использовать эту же пиктограмму во второй раз (для другого состояния или класса), достаточно просто указать ее имя. По умолчанию пиктограммы тревог добавляются в **Глобальный пул изображений**, расположенный на **Панели POU**.

В нашем проекте мы будем использовать класс **Error** для тревог о выходе температуры за **аварийные уставки** и **Info** – за **границы гистерезиса**. Их настройки приведены на рис. 7.132 и рис. 7.133.

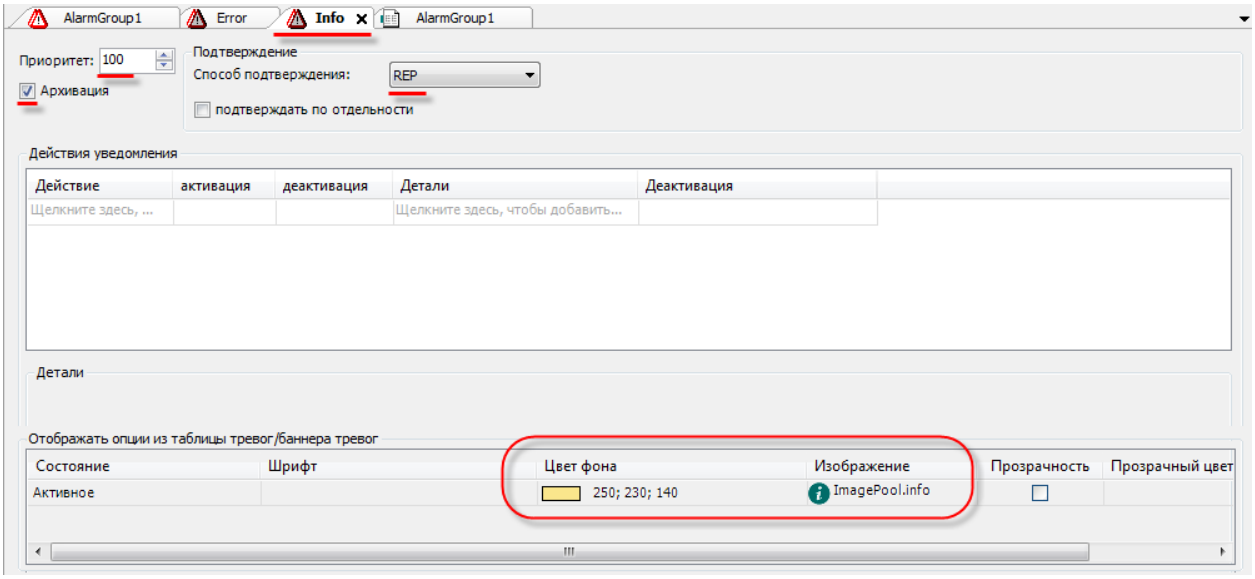


Рис. 7.133. Настройки класса тревог Info

7.6.3. Создание группы тревог

Настроим ***созданную нами*** в п 7.6.1 **группу тревог** с названием **AlarmGroup1**:

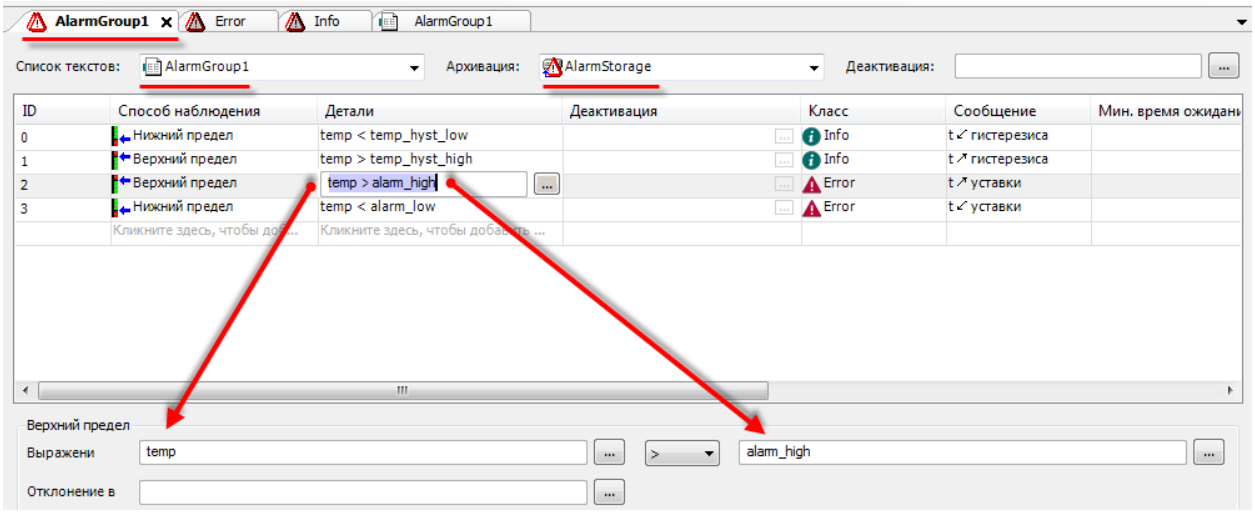


Рис. 7.134. Настройка группы тревог **AlarmGroup1**

Для группы можно указать используемый ею **Список текстов**, **Хранилище** и логическую переменную в поле **Деактивация**, которая включает (при значении TRUE) или отключает (при значении FALSE) возможность активации тревог этой группы.

Добавить новую тревогу можно двойным нажатием **ЛКМ** по ячейке столбца **Способ наблюдения**. После того, как будет выбран способ наблюдения для новой тревоги, автоматически сформируется ее **ID** и появятся подсказки по заполнению оставшихся полей. Поля, еще не заполненные надлежащим образом, помечаются иконкой .

Тревога обладает следующими настройками:

1. **ID** – идентификатор, который автоматически присваивается тревоге при ее создании. Этот ID соответствует номеру, используемому в списке текстов тревог. Его можно изменить, однако он должен оставаться уникальным в группе тревог. При изменении ID в таблице, он автоматически обновляется и в списке текстов, и наоборот;

2. **Способ наблюдения** определяет условие появления тревог. Описание доступных способов приведено в табл.8:

Табл. 8. Способы наблюдения тревоги

Способ наблюдения тревоги	Возможные настройки
Дискретный	**Выражение**: В левой части введите наблюдаемое выражение, в правой части - выражение, с которым вы хотите его сравнивать (предел); посередине выберите нужный оператор сравнения (= или <>).
Верхний предел	**Выражение**: то же, что и для типа «Дискретный» (см. выше), но с операторами сравнения > или >= и возможным параметром **Отклонение в %**[1].
Нижний предел	**Выражение**: то же, что и для типа «Дискретный» (см. выше), но с операторами сравнения < или <= и возможным параметром **Отклонение в %**[1].
Внутренний диапазон	**Выражение**: введите наблюдаемое выражение. **Область**: тревожная ситуация возникнет, как только наблюдаемое выражение примет значение из заданного диапазона. Слева введите выражение, определяющее нижнюю границу области, справа - верхнюю границу. Наблюдаемое выражение будет отображено в поле посередине. Выберите нужные операторы сравнения, а так же задайте при желании **Отклонение в %**[1].
Внешний диапазон	**Выражение**: введите наблюдаемое выражение. **Область**: тревожная ситуация возникнет, как только наблюдаемое выражение примет значение вне заданного диапазона. Слева введите выражение, определяющее нижнюю границу области, справа - верхнюю границу. Наблюдаемое выражение будет отображено в поле посередине. Выберите нужные операторы сравнения, а также задайте при желании **Отклонение в %**[1].
Изменение	**Выражение**: введите наблюдаемое выражение. Тревожная ситуация возникнет, как только его значение изменится.
Событие	В этом случае состояние тревоги переключается через приложение с помощью функций библиотеки **AlarmManager.library**.

[1] **Отклонение в %**: если задан этот параметр, то тревожная ситуация будет иметь место до тех пор, пока не будет достигнуто определенное отклонение от указанного предельного значения. Размер отклонения задается в процентах (%) от предельного значения.

Пример: Верхний предел: «i_temp >= 30», Отклонение: «10%». Как только значение переменной i_temp достигнет или превысит 30, возникнет тревожная ситуация. Пока значение не упадет до 27, сообщение о тревоге будет сохраняться.

3. **Детали** отображают текущую конфигурацию тревоги;

4. **Деактивация** позволяет указать переменную, значение которой будет влиять на исчезновение данной тревоги;

5. В столбце **Класс** указывается **класс тревог**, к которому принадлежит данная тревога;

6. В столбце **Сообщение** можно указать текст сообщения, которое будет отображаться в **Таблице тревог** при активации тревоги. Этот текст автоматически

добавляется в **Список текстов тревог**, заданный для группы. Помимо фиксированного текста, можно использовать следующие заполнители:

Табл. 9. Заполнители сообщений **Таблицы тревог**

Заместитель	Значение
DATE	Дата перехода тревоги в текущее состояние
TIME	Время последнего изменения состояния тревоги
EXPRESSION	Выражение (задается в настройках тревоги), переключившее тревогу из одного состояния в другое
PRIORITY	Приоритет тревоги (задается в настройках класса тревог)
TRIGGERVALUE[1]	Значение, вызвавшее тревогу
ALARMID	ID тревоги (задается в настройках тревоги)
CLASS	Имя класса тревог (задается в настройках тревоги)
CURRENTVALUE[1]	Текущее значение наблюдаемой переменной
LATCH1, LATCH2[1]	Значение первой и второй триггерной переменной (см. ниже)
ALARM	TRUE, если состояние тревоги «активное», FALSE в любом другом случае
STATE	Состояние тревоги: 0 - «отсутствие тревоги», 1 - «активное», 2 - «ожидание подтверждения», 3 - «активное, подтверждено»
[1] Для TRIGGERVALUE, CURRENTVALUE, LATCH1 и LATCH2 можно также использовать форматирование, например, CURRENTVALUE %2.2f	

7. **Мин. время ожидания** определяет время задержки активации тревоги;

8. **Первая и вторая триггерная** переменная используются при необходимости записи значений при активации тревоги. Триггерная переменная не должна быть массивом (в т.ч. строкой).

9. В последнем столбце можно указать существующую **тревогу с более высоким приоритетом** по отношению к создаваемой. В этом случае при одновременном наступлении двух тревог будет автоматически подтверждаться текущая. Это позволяет выполнять двухэтапную обработку тревоги, поскольку тревога с меньшим приоритетом перекрывается тревогой с большим приоритетом.

Пример для системы контроля температуры: вы задаете тревогу с меньшим приоритетом (например, 10) для ***предупреждения*** о температуре, превышающей 30 °C. Предварительно вы уже задали другую тревогу с более высоким приоритетом (например, 1), которая срабатывает при достижении температуры 50 °C (***критическое состояние***). Эту критическую тревогу можно ввести здесь, в конфигурации ***предупреждающей*** тревоги как «Тревогу с большим приоритетом». При этом существующая ***предупреждающая*** тревога будет автоматически подтверждаться при срабатывании ***критической*** тревоги.

Настроим группу тревог **AlarmGroup1** в соответствии с рис. 7.134.

7.6.4. Хранилище тревог и Список текстов

Компонент **Хранилище тревог** содержит настройки хранения файла тревог.

Файл, в который записывается информация о тревогах, хранится во **внутренней** памяти контроллера (возможность записи на USB накопитель ***отсутствует***). Пользователь не может изменять его имя, поскольку оно автоматически формируется из имени приложения следующим образом: **<имя приложения>.<имя хранилища тревог>.sqlite**. Использование файла записи определяется для классов (галочка **Архивация**) и групп тревог (указание **Хранилища**).

Хранилище тревог имеет следующие настройки:

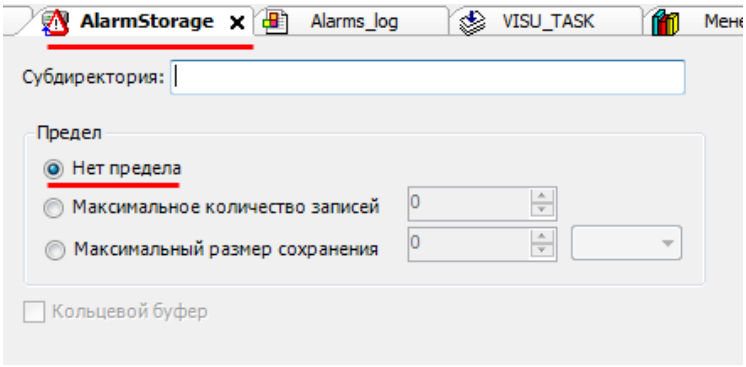


Рис. 7.135. Внешний вид **Хранилища тревог**

1. **Субдиректория** – это папка контроллера, расположенная по адресу mnt/ufs/root/CoDeSysSP_wrk, где будет сохраняться история тревог; по умолчанию (при отсутствии названия папки) файл с историей будет сохраняться в корне папки mnt/ufs/root/CoDeSysSP_wrk.

2. **Предел** позволяет ограничить количество записей в базе данных – либо по максимальному количеству, либо по размеру файла записи. При превышении ограничения старые записи начнут удаляться (кольцевой буфер).

При выполнении **заводского сброса** файл записи удаляется.

Список текстов тревог представляет собой набор текстовых сообщений, используемых для отображения в **Таблице тревог**.

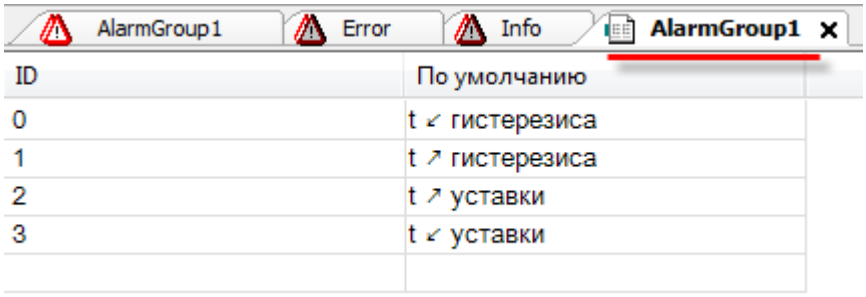


Рис. 7.136. Внешний вид **Списка текстов тревог**

7.7. Настройка задач

Задача определяет **приоритет** компонента, а также **тип выполнения** и его настройки. Компонент **Конфигурация задач** автоматически добавляется в проект при его создании. Наш проект в данный момент содержит четыре задачи:

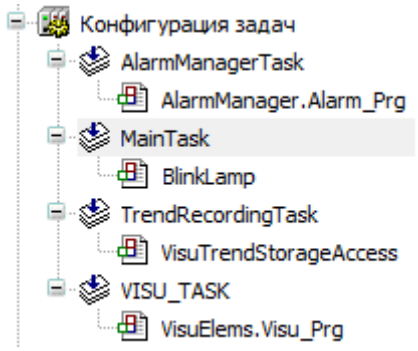


Рис. 7.137. Компонент **Конфигурация задач**

1. **AlarmManagerTask** была автоматически создана при добавлении в проект **Конфигуратора тревог** (см. п. 7.6);

2. **MainTask** была автоматически создана вместе с проектом и содержала программу PLC_PRG, которую мы переименовали в **BlinkLamp** (см. п. 7.4.3)

3. **TrendRecordingTas**k была автоматически создана при добавлении в визуализацию элемента **Тренд** (см. п. 7.3.3);

4. **Visu_Task** была автоматически создана при добавлении в проект первой визуализации (см. п. 6).

Переименуем задачу **MainTask** (с помощью двойного нажатия **ЛКМ** на название задачи или одиночного нажатия **ПКМ** и выбора пункта Refactoring) в **BlinkTask**. Теперь создадим новую задачу с названием **MainTask**:

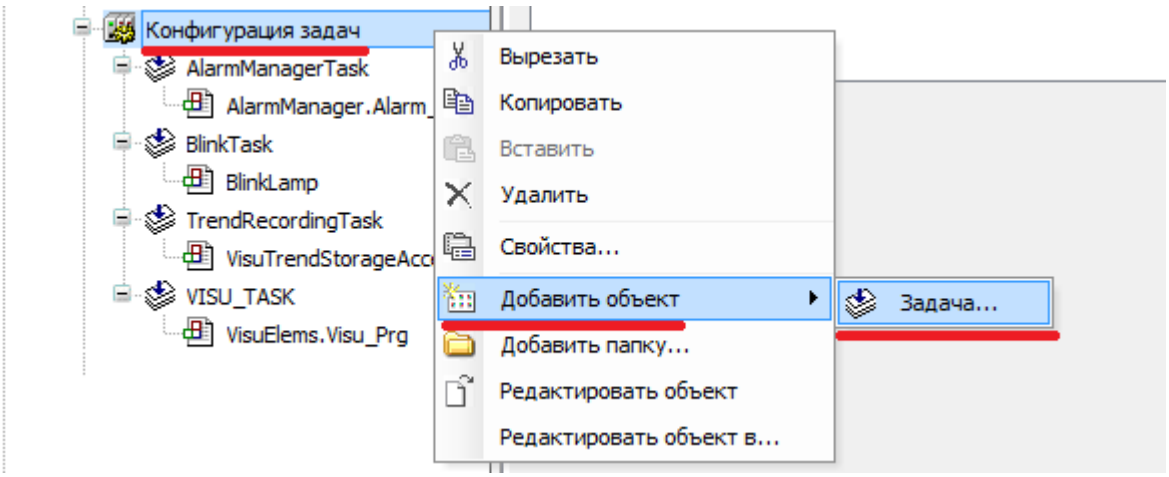


Рис. 7.138. Создание новой задачи

Выберем программу **MainPrg**, зажмем **ЛКМ** и, ***не отпуская ее***, перетащим программу под соответствующую задачу. В итоге компонент **Конфигурация задач** будет выглядеть следующим образом:

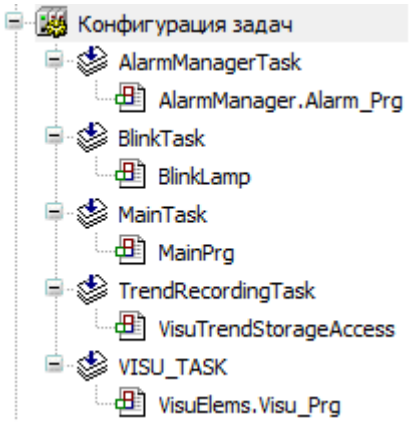


Рис. 7.139. Компонент Конфигурация задач после окончательной настройки

Рассмотрим настройки задачи на примере **BlinkTask**:

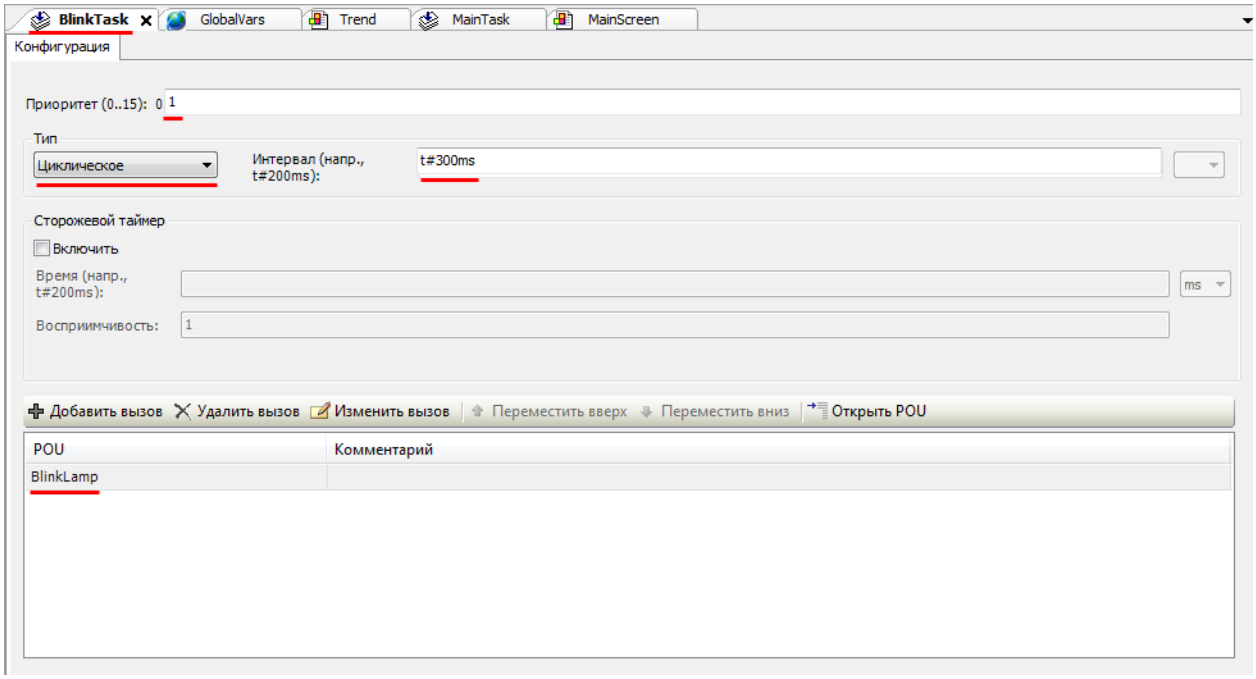


Рис. 7.140. Настройки задачи **BlinkTask**

1. **Приоритет** характеризует порядок выполнения задачи. При прочих равных условиях первой будет выполняться задача с меньшим приоритетом. Самый низкий приоритет – 0, самый высокий – 31;

2. **Тип** определяет условие вызова задачи:

Табл. 10. Типы выполнения задач

Тип	Условие вызова задачи
Циклическая	Задача вызывается циклически через заданный интервал времени (иными словами, этот интервал определяет время одного цикла задачи)
Свободное выполнение	Задача вызывается сразу же после своего окончания, в непрерывном цикле без задания каких-либо интервалов
Статус	Задача вызывается, если логическая переменная, заданная в поле **Событие**, имеет значение TRUE
Событие	Задача вызывается, если логическая переменная, заданная в поле **Событие**, ***меняет*** свое значение с FALSE на TRUE

3. **Сторожевой таймер** (watchdog) позволяет прервать задачу, если время ее выполнения превысило заданный допустимый предел (время цикла);

4. **Меню вызова POU** определяет список компонентов, исполняемых в рамках данной задачи.

Настройки задач **BlinkTask** и **MainTask** приведены на рис. 7.140. и 7.141.

Для задачи **BlinkTask** мы используем интервал цикла, равный **400 мс**; это связано с тем, что блок **Blink** программы **BlinLamp** работает в собственно цикле, время которого определятся значениями его входных переменных: **TIMELOW=200 мс**, **TIMEHIGH=200 мс**. Соответственно, чтобы блок работал корректно (напомним, он используется для реализации мигающего индикатора), время цикла задачи должно соответствовать времени цикла блока. Время цикла блока определяется пользователем; для упрощения примем, что за один цикл блок генерирует один импульс и одну паузу. Соответственно, время цикла задачи = (1·200 мс + 1·200 мс) = 400 мс. Точно так же мы могли бы задаться временем цикла задачи, равным 800 мс – в этом случае в течение одного цикла блок бы генерировал два импульса и две паузы попеременно.

Для задачи **MainTask** мы используем интервал, равный **одной секунде**, т.к. в программе **MainPrg**, связанной с этой задачей, реализована эмуляция изменения температуры – т.е. при работе регулятора значение температуры будет изменяться на 3 градуса Цельсия каждый цикл. Чтобы сделать процесс изменения температуры более наглядным, зададимся соответствующим временем цикла.

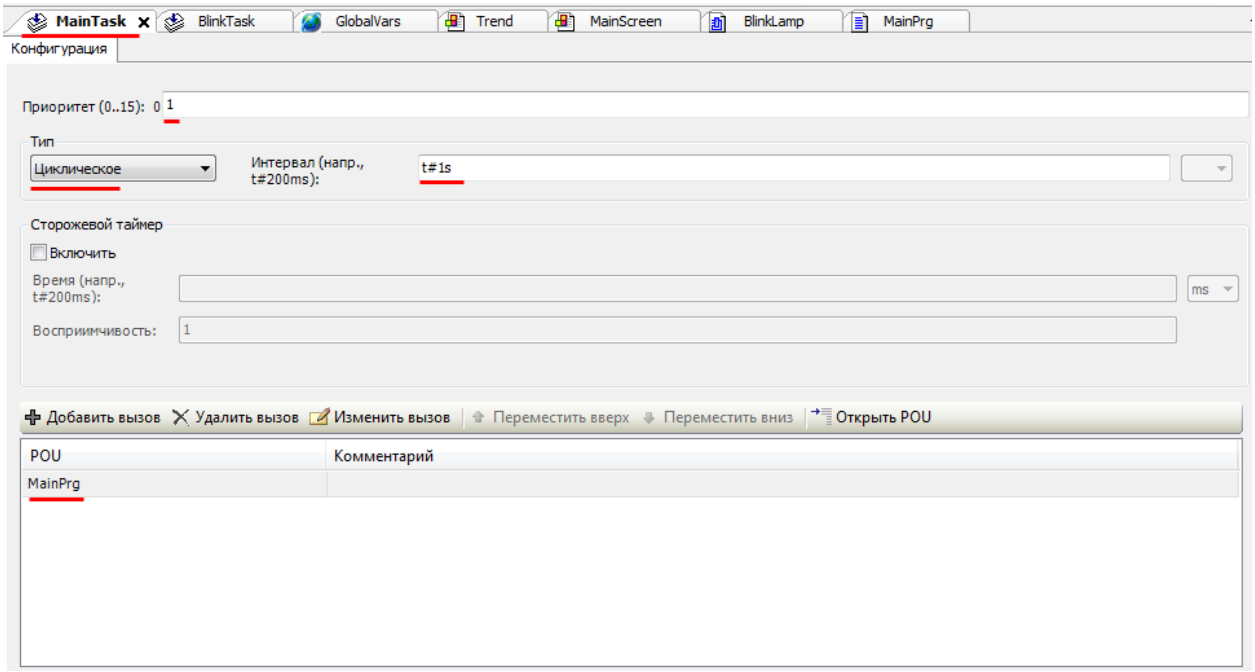
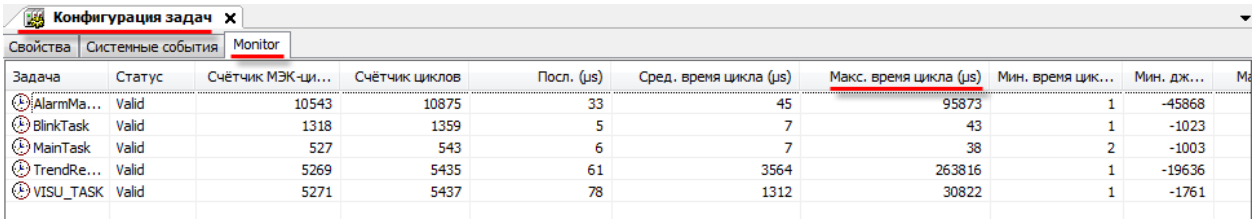


Рис. 7.141. Настройки задачи **MainTask**

Во время работы проекта становится активной (заполняется данными)вкладка **Монитор** компонента **Конфигурация задач**.



Задача	Статус	Счётчик МЭК-ци...	Счётчик циклов	Посл. (µs)	Сред. время цикла (µs)	Макс. время цикла (µs)	Мин. время цик...	Мин. дж...	Ма
AlarmMa...	Valid	10543	10875	33	45	95873	1	-45868	
BlinkTask	Valid	1318	1359	5	7	43	1	-1023	
MainTask	Valid	527	543	6	7	38	2	-1003	
TrendRe...	Valid	5269	5435	61	3564	263816	1	-19636	
VISU_TASK	Valid	5271	5437	78	1312	30822	1	-1761	

Рис. 7.142. Вкладка **Монитор** Конфигурации задач

Эта вкладка содержит статистическую информацию о работе проекта. В частности, рекомендуется обратить внимание на параметр **Макс. время цикла** – его значение ***не должно превышать*** время цикла, заданное для конкретной задачи (если задача исполняется циклически). В противном случае рекомендуется либо увеличить время цикла задачи, либо «облегчить» содержимое компонентов, привязанных к ней.

Напоминаем, что при необходимости ***прервать*** задачу, если она не успела выполниться за время цикла, можно воспользоваться **Сторожевым таймером** (**watchdog),** который включается в настройках задачи.

7.8. Настройка обмена данными по протоколу Modbus RTU

В процессе разработки

В данный момент можно воспользоваться руководством Настройка обмена по протоколу Modbus в среде CODESYS V3.5.

8. Компиляция и загрузка проекта

8.1. Компиляция проекта

После выполнения действий, описанных в п. 7, мы имеем готовый пользовательский проект. Перед тем, как загрузить его в контроллер, необходимо произвести **компиляцию** с помощью соответствующей вкладки на **Панели инструментов**:

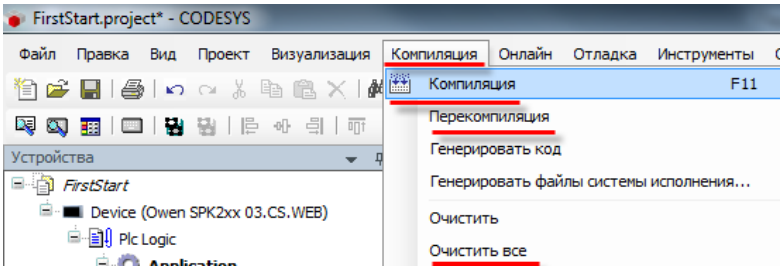


Рис. 8.1. Команды вкладки **Компиляция**

Компиляция представляет собой проверку синтаксиса пользовательского приложения. Команда **Перекомпиляция** позволяет выполнить компиляцию ранее скомпилированного проекта.

После выполнения компиляции в папке, где хранится проект, создаются файлы, содержащие информацию о результатах компиляции. При необходимости удалить информацию о результатах предыдущих компиляций можно с помощью команды **Очистить все**.

В целом, ***рекомендуется*** перед загрузкой проекта в контроллер (и после любых значительных изменений в проекте) выполнять последовательно команды **Очистить все** и **Перекомпиляция**.

Информация об ошибках и предупреждениях, возникших в ходе компиляции, отображается на соответствующей панели:

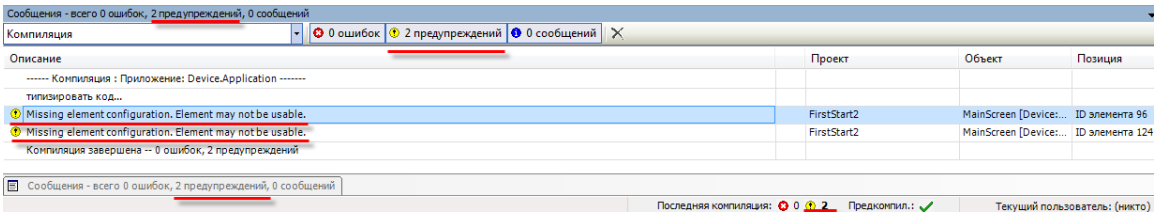


Рис. 8.2. Панель сообщений компиляции

В данном случае мы видим два предупреждения – они связаны с тем, что к серым индикаторам на экране **MainScreen** не привязаны переменные (т.к. нам не нужны светящиеся серые лампы).

Проект, скомпилированный с **предупреждениями**, ***можно*** загрузить в контроллер; проект, скомпилированный с **ошибками,** загрузить в контроллер ***нельзя***.

8.2. Загрузка проекта в контроллер

После компиляции, ***в ходе которой не возникло ошибок***, мы можем загрузить проект в контроллер. Загрузить проект можно либо с компьютера, который подключен к контроллеру, либо с **USB накопителя**. Процесс загрузки проекта с накопителя описан в Руководстве по эксплуатации.

Рассмотрим процесс загрузки проекта в контроллер с пользовательского ПК. ***Предварительно должна быть настроена связь между контроллером и компьютером*** (см. п. 5). Ранее мы уже загружали пустой проект в **оперативную память** контроллера (см. п. 6). Напомним, что информация из оперативной памяти ***удаляется*** после выключения контроллера. Поскольку проект уже готов, загрузим его во **flash-память**, содержимое которой ***сохраняется*** после выключения контроллера.

Выберем в меню **Онлайн** команду **Логин** для подключения к контроллеру:

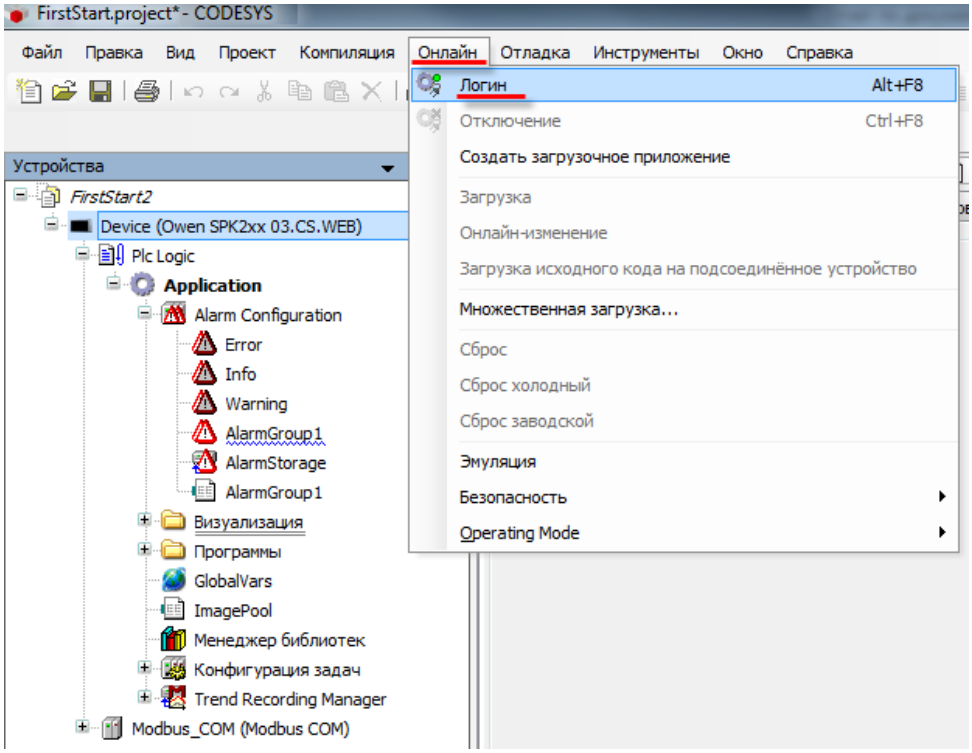


Рис. 8.3. Команда **Логин** (подключение к контроллеру)

Так как в контроллер уже загружен пустой проект (см. п.3), то появится следующее информационное сообщение:

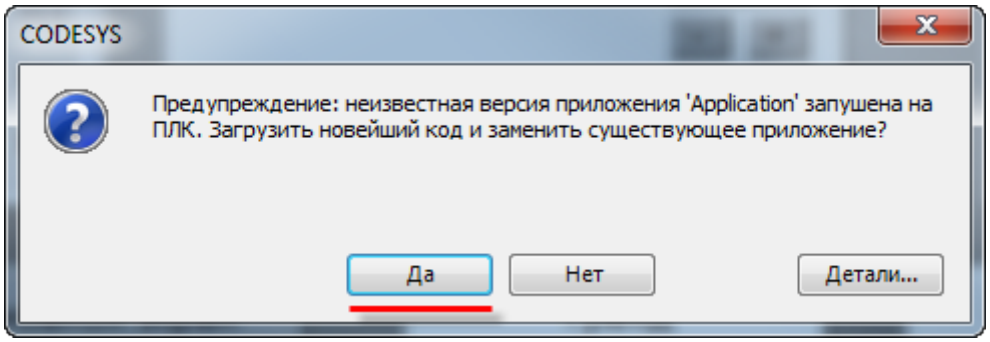


Рис. 8.4. Информационное окно загрузки проекта (при наличии в контроллере другого проекта)

Нажмем **Да**. В результате пустой проект будет ***удален***, а новый (созданный в п. 7) записан в **оперативную память** контроллера.

Для загрузки проекта во **flash-память** нажмем кнопку **Создать загрузочное приложение** в меню **Онлайн**:

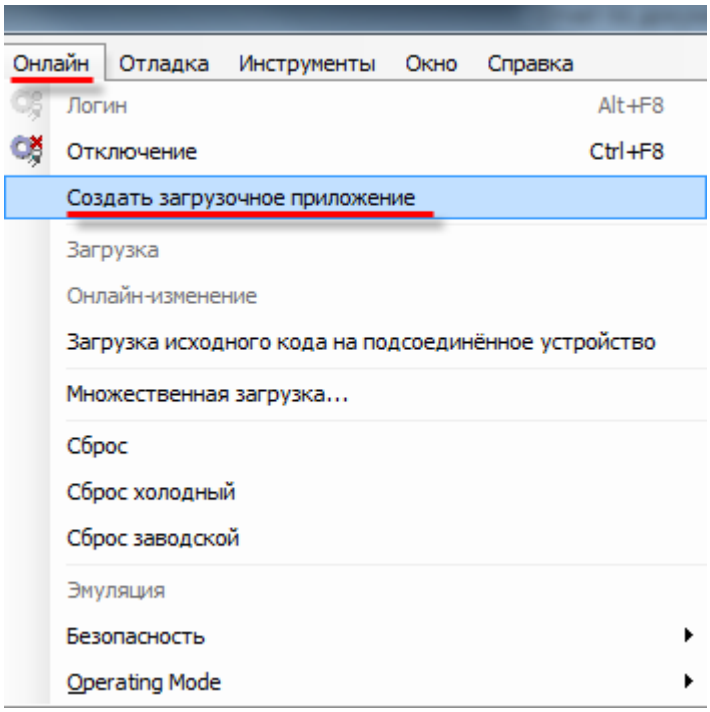


Рис. 8.5. Создание загрузочного приложения (загрузка проекта во flash-память контроллера)

Также рекомендуется выполнить **Загрузку исходного кода на подсоединенное устройство** – это позволит при необходимости ***выгрузить*** проект CODESYS из памяти контроллера:

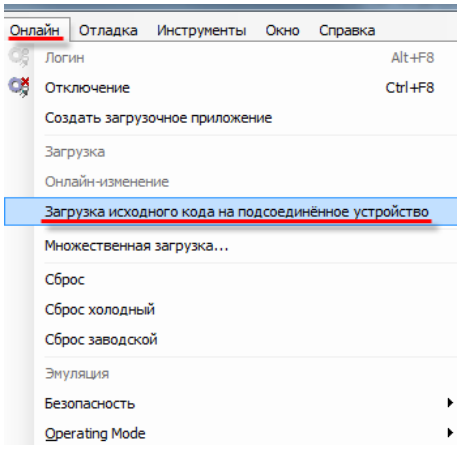


Рис. 8.6. Загрузка исходного кода на подсоединенное устройство

Запустим проект (меню **Отладка – Старт**):

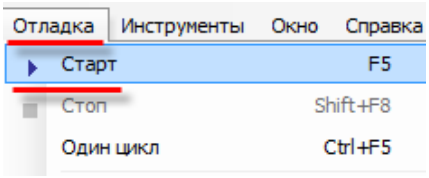


Рис. 8.7. Запуск проекта

Теперь наш проект загружен во **flash-память** контроллера и ***запущен***; при перезагрузке контроллера проект ***сохранится*** в памяти контроллера и будет ***автоматически перезапущен***.

При необходимости ***удалить проект*** из контроллера, следует воспользоваться командой **Сброс заводской** из меню **Отладка**:

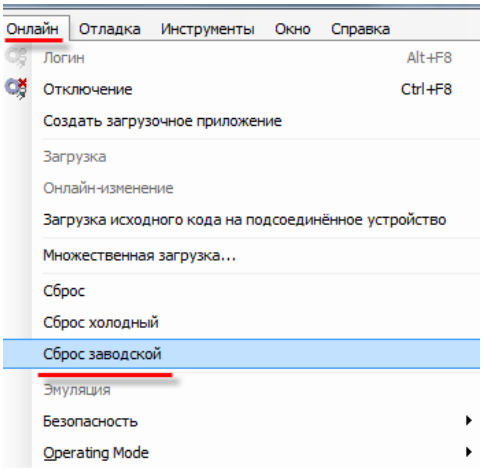


Рис. 8.8. Удаление проекта из контроллера

Команды типа **Сброс** выполняют следующие действия:

1. **Сброс** – заново инициализирует все переменные, ***кроме*** энергонезависимых (retain);

2. **Сброс холодный** – заново инициализирует все переменные, ***включая*** энергонезависимые (retain);

3. ***Сброс заводской*** – заново инициализирует все переменные, включая энергонезависимые (retain), ***после чего удаляет проект*** из контроллера.

В процессе разработки проекта зачастую приходится вносить в него изменения и ***заново загружать*** в контроллер; в этом случае при загрузке можно увидеть следующее информационное окно:

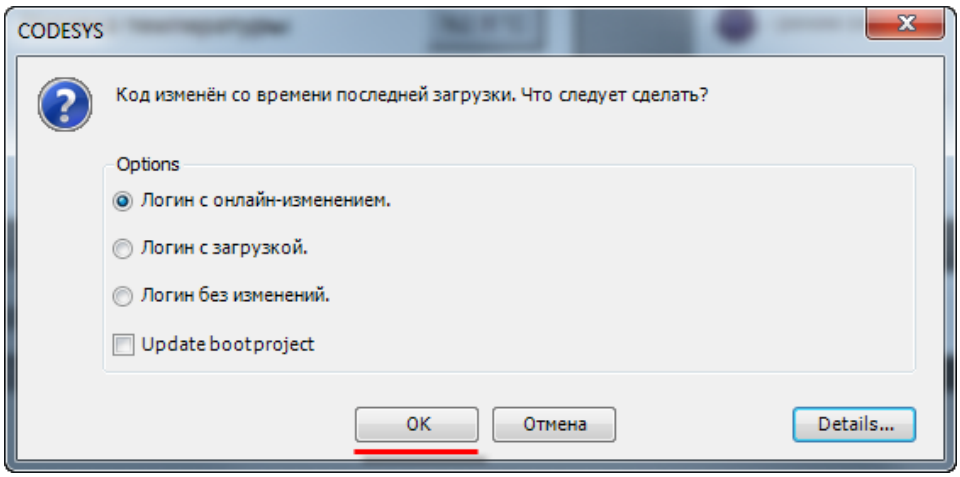


Рис. 8.9. Диалоговое окно обновления проекта

1. Команда **Логин с онлайн-изменением** загружает в **оперативную память** контроллера ***только те части проекта***, которые подверглись изменениям с момента последней загрузки; процедура онлайн-изменения опирается на использовании **информации компиляции**, поэтому невозможна после выполнения команды **Очистить** из меню **Компиляция;**

2. Команда **Логин с загрузкой** загружает проект в оперативную память контроллера (стандартная процедура загрузки);

3. Команда **Логин без изменений** осуществляет подключение к контроллеру ***без загрузки проекта***.

4. При заполненном чекбоксе **Update bootproject** при выполнении любой из вышеописанных команд происходит обновление проекта во flash-памяти контроллера.

В большинстве случаев ***рекомендуется*** использовать команду **Логин с загрузкой**.

9. Графический дизайн проекта

При разработке экранов визуализации (см. п. 7.3), мы использовали только графические примитивы, входящие в состав **CODESYS**. Но при разработке сложного и эргономичного интерфейса оператора может возникнуть необходимость в использовании дополнительных изображений; для этого можно воспользоваться элементом **Переключатель изображений**, расположенном на вкладке **Индикаторы/Переключатели/Изображения**.

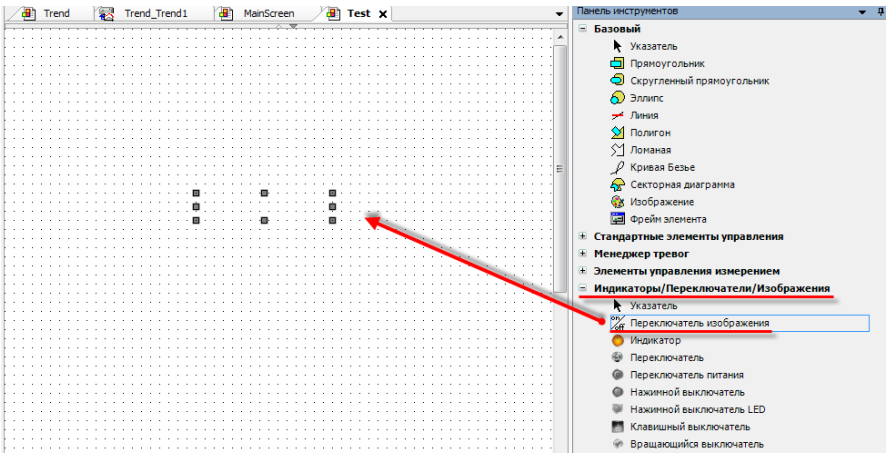


Рис. 9.1. Добавление элемента **Переключатель изображения**

Элемент имеет следующие настройки:

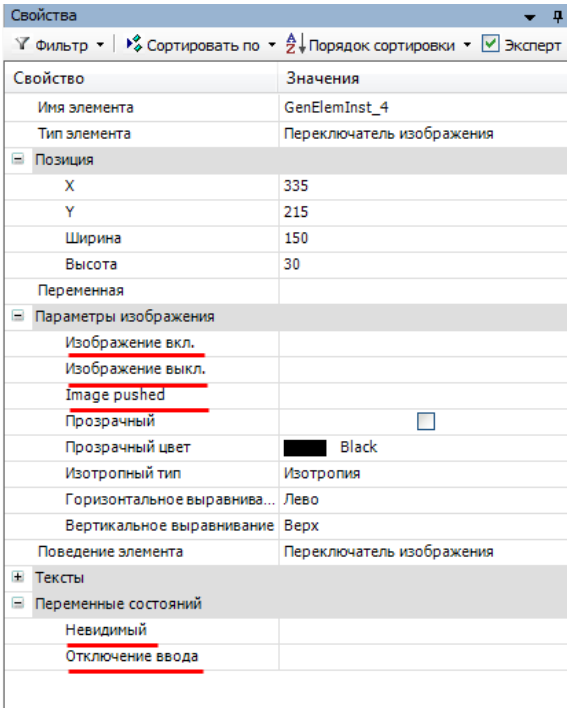


Рис. 9.2. Настройки элемента **Переключатель изображения**

Элемент работает следующим образом: если привязанная к нему логическая переменная принимает значение **TRUE**, то отображается изображение, указанное в поле **Изображение вкл.;** если переменная принимает значение **FALSE** – отображается изображение, указанное в поле **Изображение выкл**. Предварительно оба изображения должны быть добавлены в проект через **Пул изображений** (см. п. 7.3.5).

По нажатию на элемент, переменная меняет свое состояние (с TRUE на FALSE или наоборот); при этом элемент на короткое время отображается изображение, заданное в поле **Image pushed**.

Как и для других элементов, для **Переключателя изображений** можно настроить **Невидимость** и **Отключение ввода**.

С помощью этого элемента для проекта, разработанного нами в п. 7, был создан стильный и эргономичный дизайн. Также на этом этапе был добавлен стартовый экран проекта; поскольку использованные на нем элементы (текстовые поля и кнопка перехода) уже рассмотрены в п. 7, то процесс создания этого экрана описываться не будет.

В приобретенный Вами контроллер по умолчанию загружена именно дизайнерская версия проекта.

Экраны визуализации в ней выглядят следующим образом:

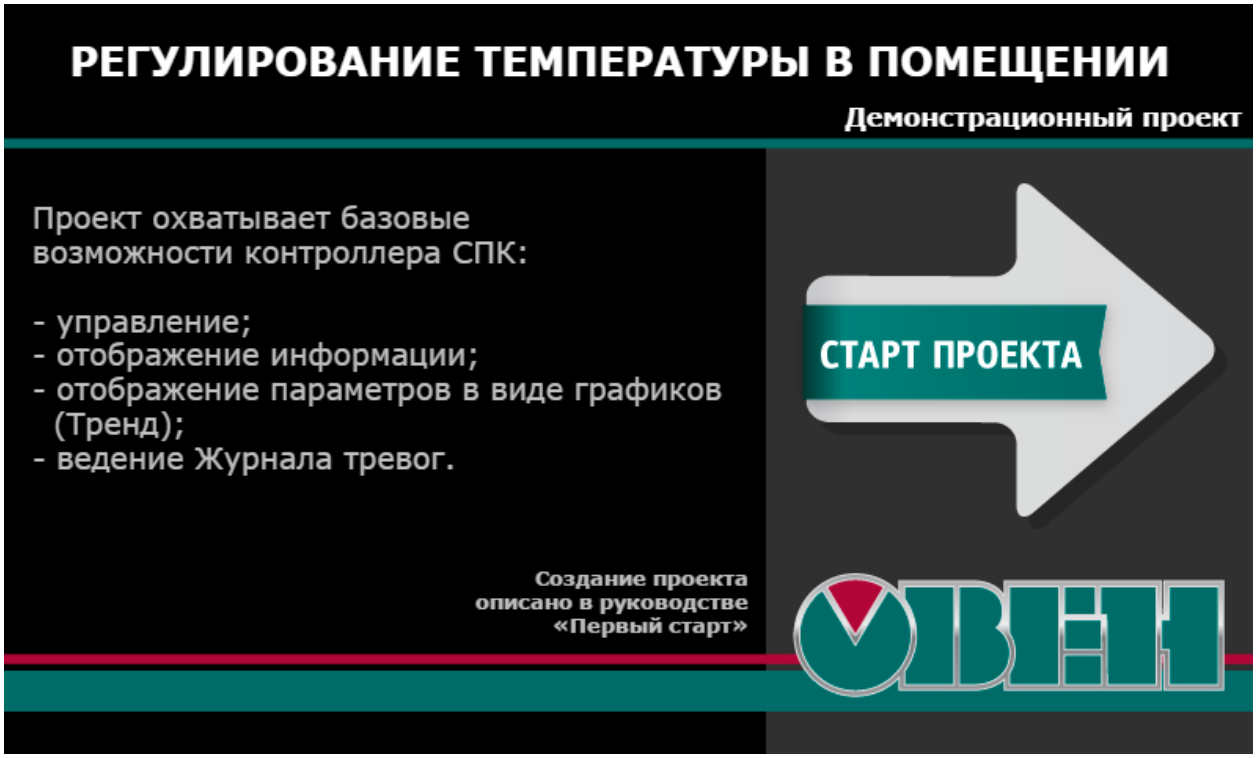


Рис. 9.3. Экран **Start** (дизайнерская версия проекта)

Рис. 9.4. Экран **MainScreen** (дизайнерская версия проекта)

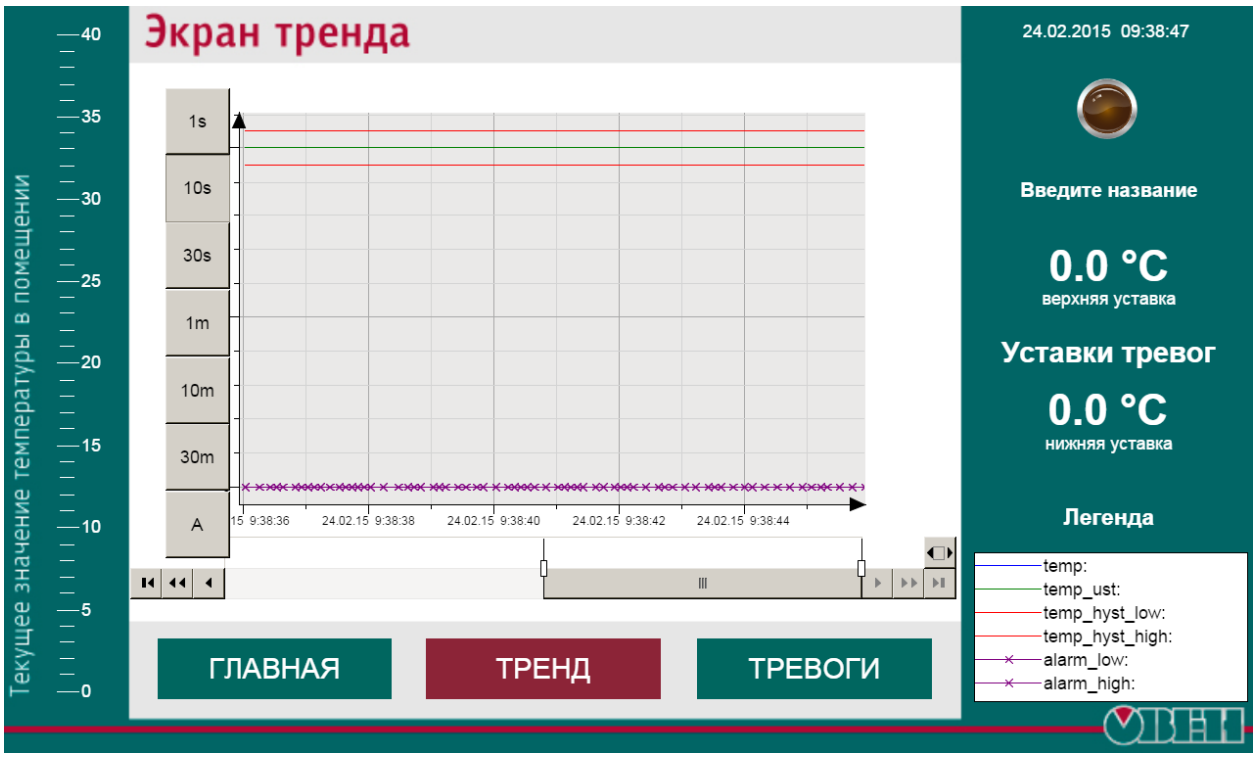


Рис. 9.5. Экран **Trend** (дизайнерская версия проекта)

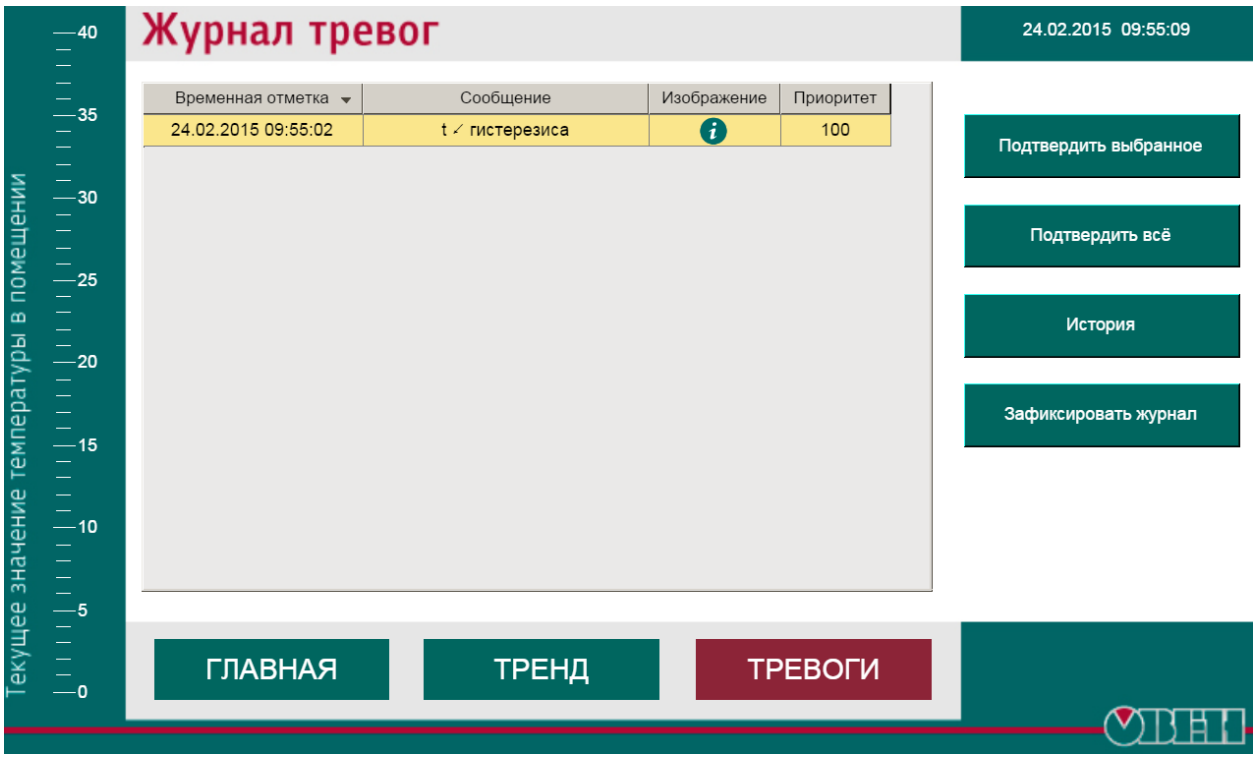


Рис. 9.6. Экран **Alarms_log** (дизайнерская версия проекта)

10. Работа с демонстрационным проектом

Итак, после выполнения п.8, пользовательский проект загружен в контроллер. После включения контроллера, он будет автоматически запущен и на его дисплее будет отображен стартовый экран визуализации (стартовый экран загруженного в контроллер по умолчанию проекта приведен на рис. 10.1).

Если модель контроллера ***поддерживает*** web-визуализацию, то для того, чтобы запустить ее, необходимо в браузере (подходит любой современный браузер с поддержкой HTML5), открыть страницу:

http://<IP-адрес контроллера>:8080/<имя страницы>.htm

Для настроек по умолчанию (***как в нашем примере***) это, соответственно, страница:

http://10.0.6.10:8080/webvisu.htm

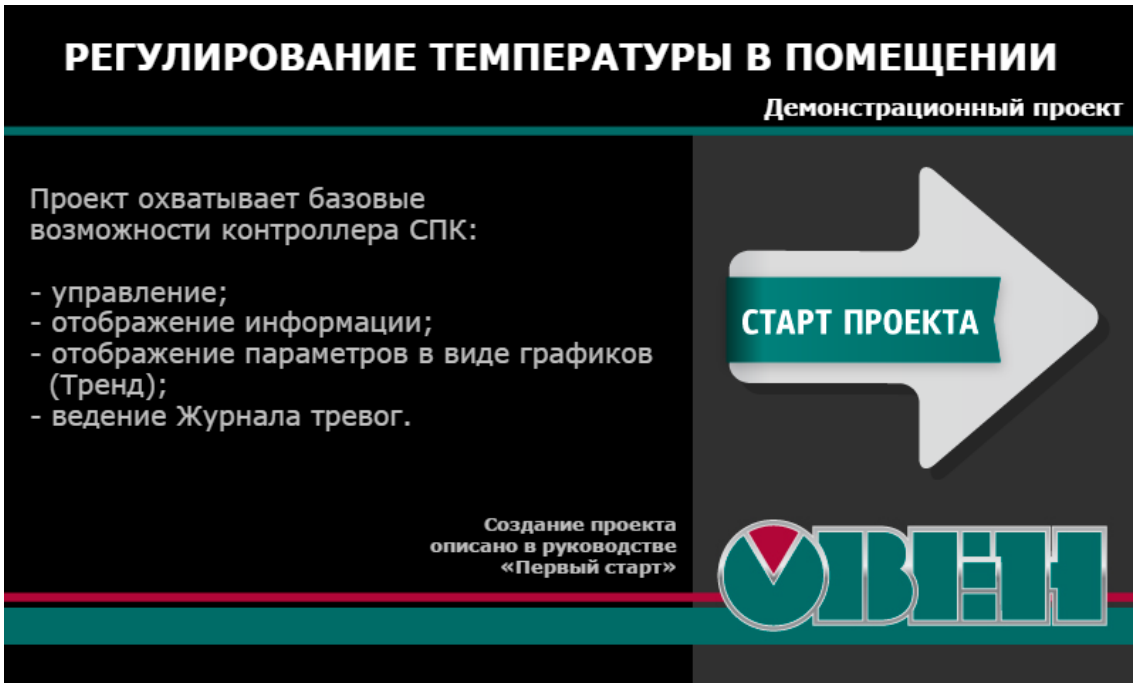


Рис. 10.1. Экран **Start** (дизайнерская версия проекта)

Нажмем кнопку **Старт проекта** для перехода на экран **Mainscreen.**

По умолчанию установлен **режим эмуляции**, уставка температуры – 20 °С, гистерезис – 5%.

На данный момент кондиционер отключен – это видно по отсутствию подсветки пиктограммы **OFF**. Введем текущее значение температуры, равное 30.5 °С:

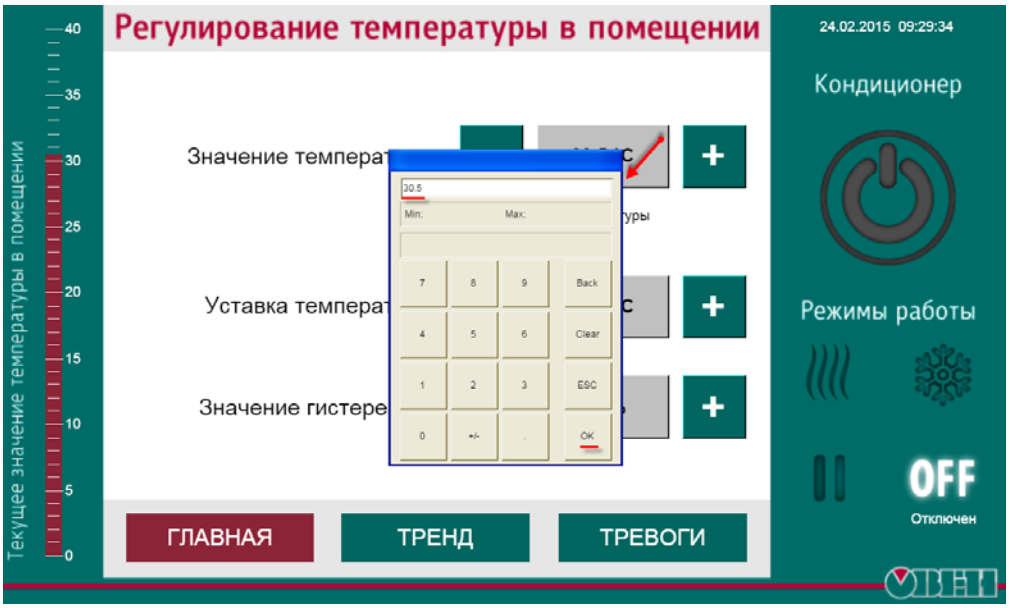


Рис. 10.2. Экран **MainScreen**, ввод значения температуры

Включим кондиционер нажатием соответствующей кнопки:

Рис. 10.3. Экран **MainScreen**, включение кондиционера

Кнопка включения кондиционера и пиктограмма режима его работы («охлаждение») загорятся, а значение температуры начнет уменьшаться на 3 градуса в секунду. Вскоре система войдет в ***режим автоколебаний*** (т.к. шаг изменения температуры больше зоны гистерезиса).

Перейдем на экран **Тренд**, нажав одноименную кнопку:

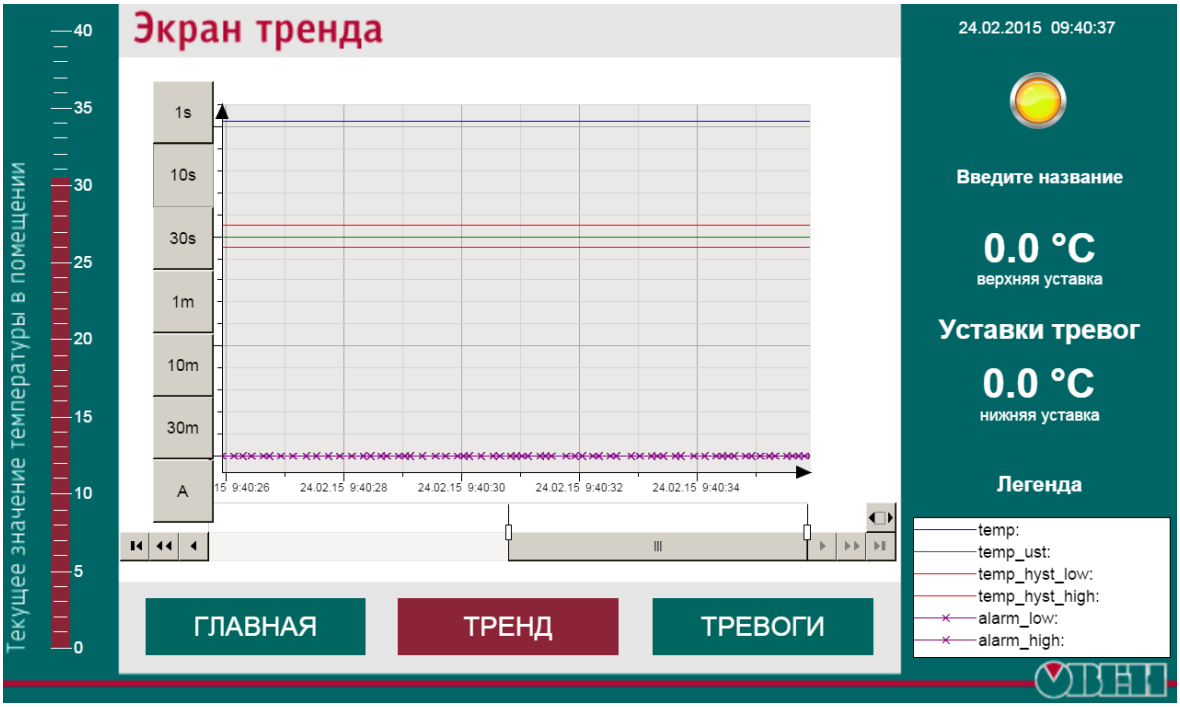


Рис. 10.4. Экран **Trend** (дизайнерская версия проекта)

По умолчанию значения обоих уставок тревог равны нулю; т.к. текущая температура колеблется в районе 20 градусов, то ***мигает*** желтая сигнальная лампа. Нажмем на поле **Введите название** и введем **ALARM**:

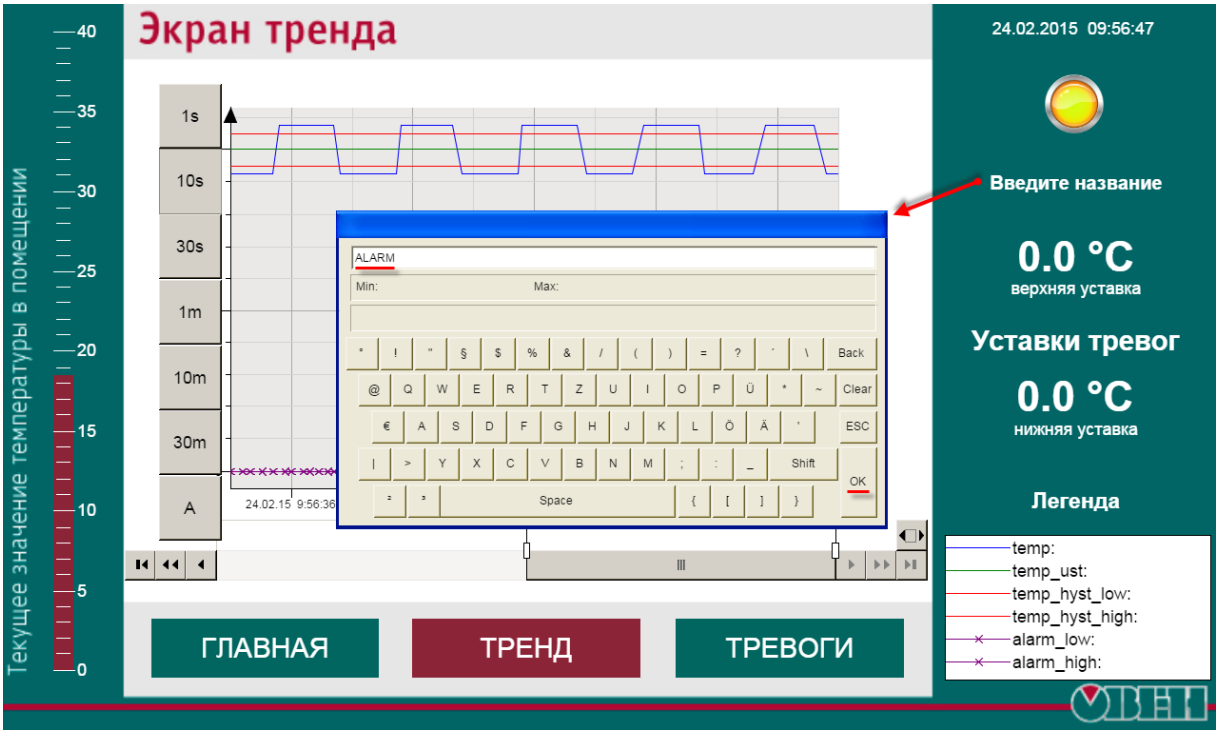


Рис. 10.5. Экран **Trend**, ввод названия лампы

Зададим уставки тревог: верхняя - 22 °С, нижняя - 18 °С.

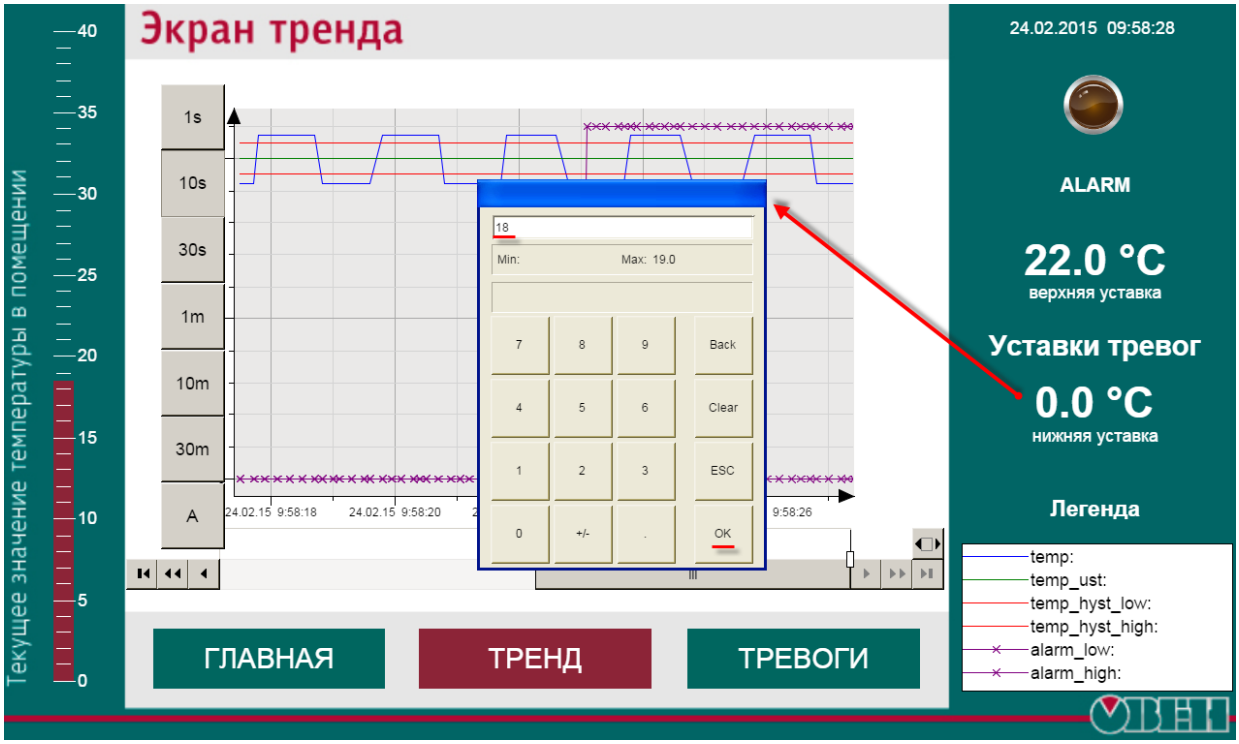


Рис. 10.6. Экран **Trend**, ввод уставок тревог

В итоге на тренде будут отображаться ***автоколебания***, которые мы создали на экране **MainScreen**:

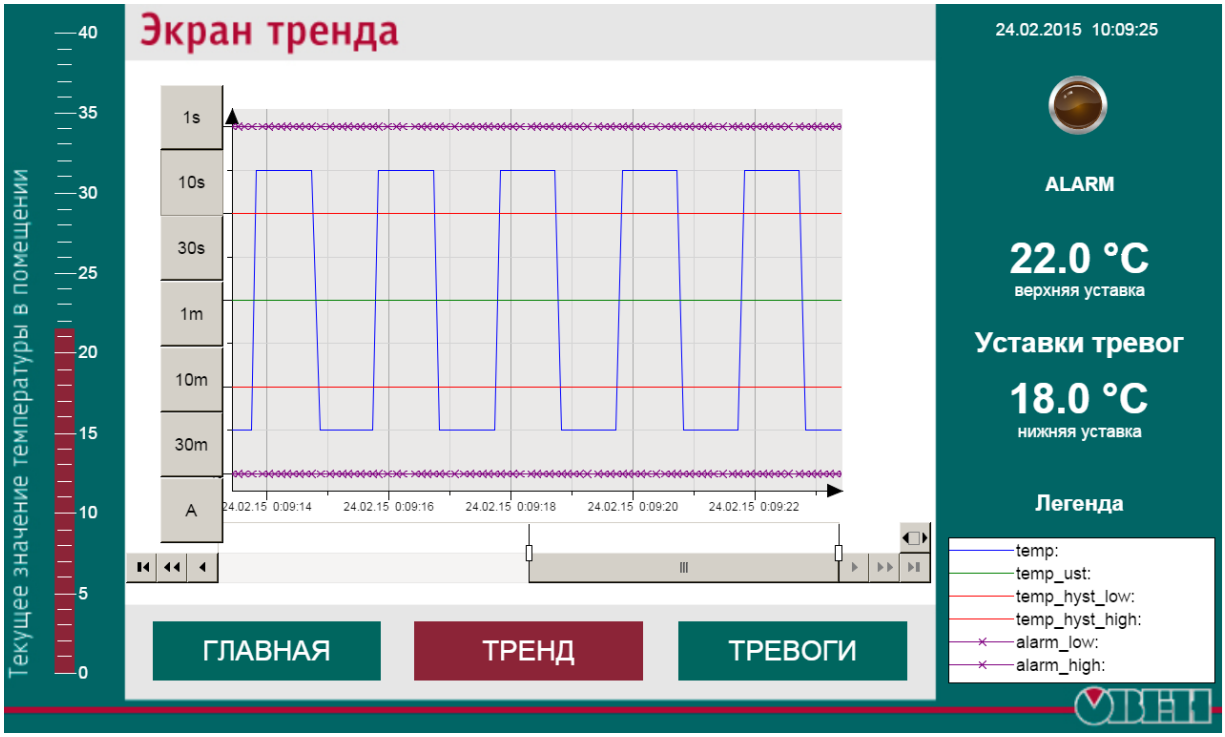


Рис. 10.7. Экран **Trend**, отображение автоколебаний

Для изменения периода времени, отображаемого на тренде, необходимо воспользоваться **левым рядом кнопок**. Для отображения истории необходимо воспользоваться **ползунком**, расположенным под трендом:

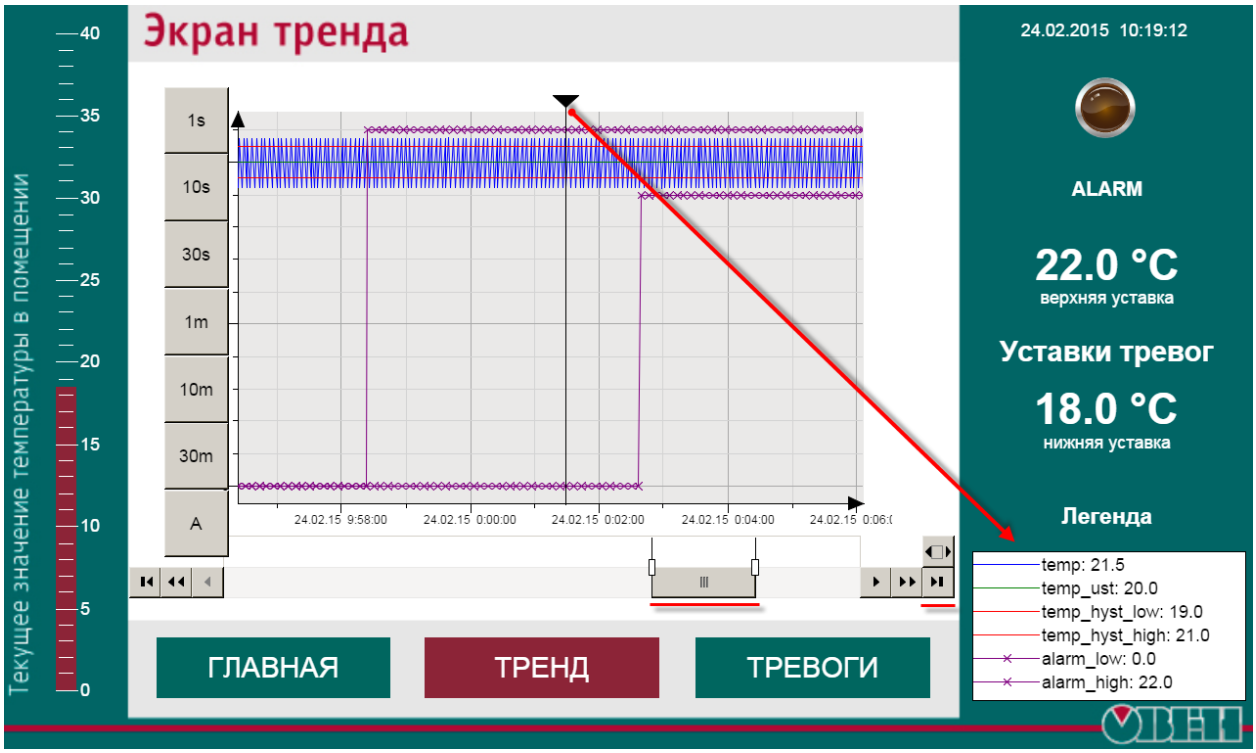


Рис. 10.8. Экран **Trend**, просмотр истории

При просмотре истории обновление тренда не происходит; также в этом режиме появляется **маркер,** который позволяет посмотреть значения переменных тренда в тот или иной момент времени (значения отображаются рядом с **легендой**).

Для того, чтобы вернуться к отображению информации в реальном времени, необходимо нажать на самую **правую нижнюю кнопку** тренда.

Напоминаем, что поскольку аварийные уставки являются **энергонезависимыми (retain)** переменными, их значения будут ***сохраняться после перезагрузки*** контроллера.

Перейдем на экран **Тревоги**, нажав одноименную кнопку:

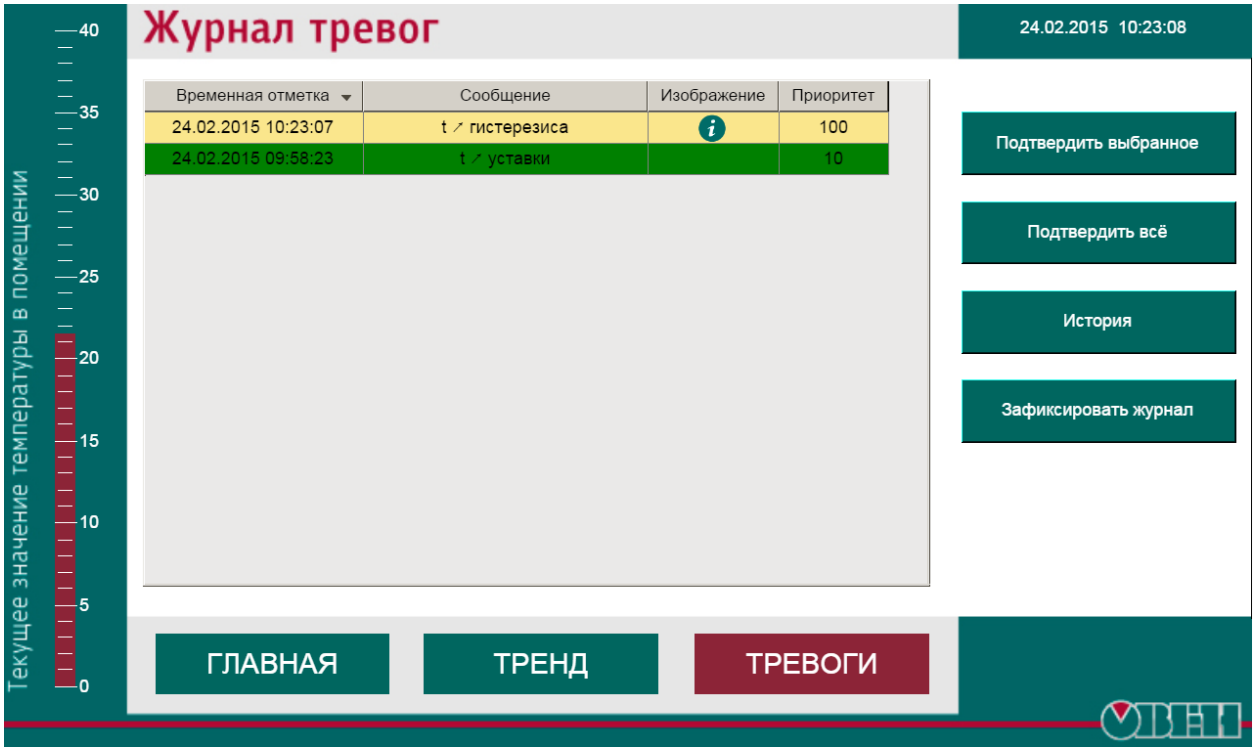


Рис. 10.9. Экран **Alarms_log** (дизайнерская версия проекта)

В **Журнале тревог** отображаются два сообщения – одно из них о выходе температуры за верхнюю/нижнюю границы гистерезиса (эти сообщения постоянно сменяют друг друга, т.к. система в режиме автоколебаний), второе – о превышении температурой аварийной уставки. Сообщение о превышении уставки сформировалось в момент задания значения температуры (т.к. по умолчанию уставка равно нулю); в данный момент температура меньше значения аварийной уставки, поэтому сообщение ожидает подтверждения (фон сообщения – зеленого цвета, пиктограмма тревоги – отсутствует).

После нажатия на кнопку **Подтвердить всё** сообщение о превышении уставки исчезнет из журнала; сообщение о выходе температуры за границы гистерезиса продолжит появляться, т.к. условие его появление продолжает выполняться.

Нажатие на кнопку **История** позволяет посмотреть историю журнала тревог:

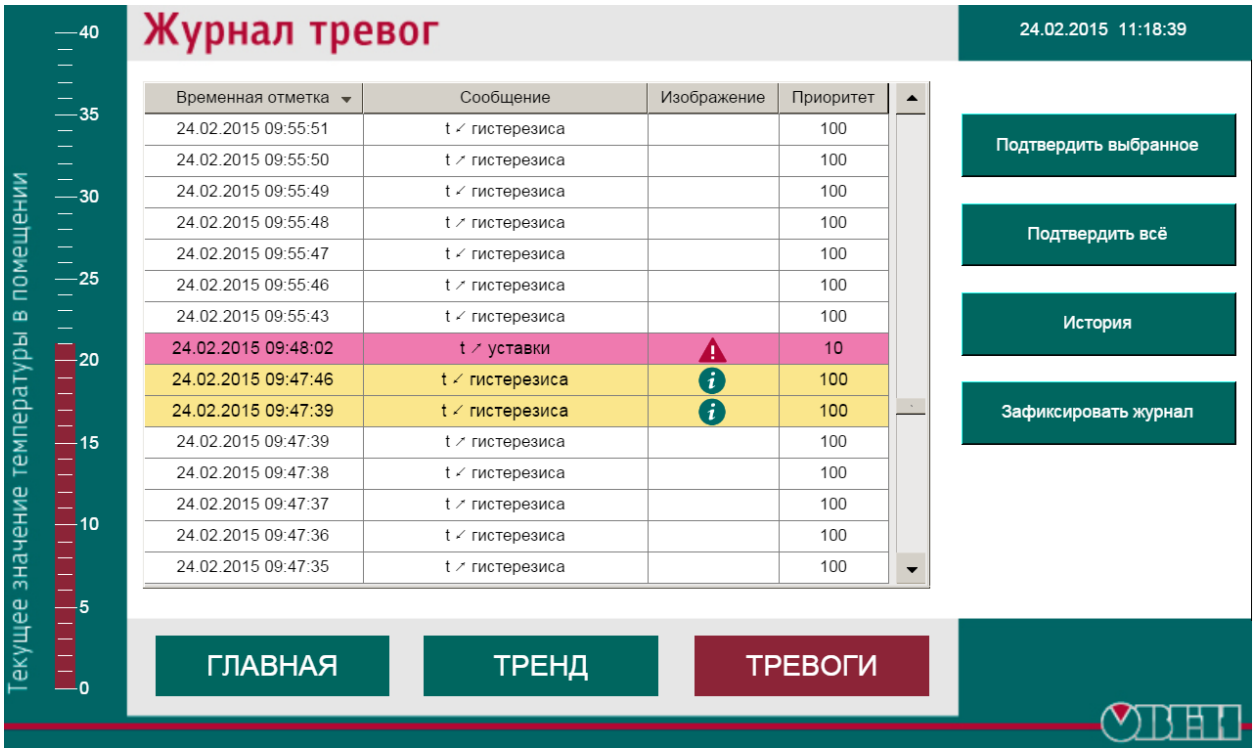


Рис. 10.10. Экран **Alarms_log** (дизайнерская версия проекта)

Для возвращения в режим реального времени, следует повторно нажать кнопку **История**.

11. Полезные ссылки

1. Руководство по эксплуатации

   контроллеров серии СПК1xx

   контроллеров серии СПК2xx

2. Инструкция по перепрошивке (см. раздел Сервисное ПО)

3. Инструкции по компонентам и примеры программ

4. Раздел СПК на сайте owen.ru

5. Раздел СПК на форуме owen.ru

6. Техническая поддержка компании ОВЕН