

# **НАСТРОЙКА ОБМЕНА ДААННЫМИ ПО ПРОТОКОЛУ HART НА КОНТРОЛЛЕРАХ СЕРИИ REGUL RX00**

**Руководство пользователя**

**DPA-302.3**

**Версия документа 1.7**

**Версия ПО 1.6.5.0**

**Июль 2021**

## История изменений руководства пользователя

Версия руководства пользователя	Описание изменения
1.6	<p>Добавлена история изменений руководства пользователя.</p> <p>Добавлены знаки с предупреждающей и поясняющей информацией.</p> <p><i>Раздел «Алгоритм работы HART-мастера»:</i> дополнено описание режима работы в зависимости от типа модуля.</p> <p>Дополнительно по тексту внесены небольшие изменения с уточняющей информацией</p>
1.7	<p>Добавлен новый раздел: «Поддержка устройств в Burst режиме».</p> <p>Дополнительно по тексту внесены небольшие изменения с уточняющей информацией</p>

## АННОТАЦИЯ

Настоящий документ содержит сведения о настройке передачи данных с применением протокола HART на промышленных логических контроллерах серии Regul RX00. Настройка осуществляется с помощью программного обеспечения Epsilon LD.

Данное руководство предназначено для эксплуатационного персонала и инженеров-проектировщиков АСУ ТП, которые должны:

- иметь, как минимум, среднее техническое образование;
- приступить к работе только после изучения данного руководства.


### Обновление информации в Руководстве

Производитель ООО «Прософт-Системы» оставляет за собой право изменять информацию в настоящем Руководстве и обязуется публиковать более новые версии с внесенными изменениями. Обновленная версия Руководства доступна для скачивания на официальном сайте Производителя: <https://www.prosoftsystems.ru/>.


Для своевременного отслеживания выхода новой версии Руководства рекомендуется оформить подписку на обновление документа. Для этого необходимо на сайте Производителя: <https://www.prosoftsystems.ru/> во вкладке «Документация» под иконками документов кликнуть на кнопку «Подписаться на обновления» и оставить свои контактные данные.

В руководстве присутствуют знаки с предупреждающей и поясняющей информацией. Каждый знак обозначает следующее:

### ПРЕДУПРЕЖДАЮЩИЕ ЗНАКИ

	<p><b>ВНИМАНИЕ!</b> Здесь следует обратить внимание на способы и приемы, которые необходимо в точности выполнять во избежание ошибок при эксплуатации или настройке.</p>
---	--

### ИНФОРМАЦИОННЫЕ ЗНАКИ

	<p><b>ИНФОРМАЦИЯ</b> Здесь следует обратить внимание на <u>важную</u> информацию</p>
---	--

## СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b> .....	<b>5</b>
Общие сведения.....	5
Перечень рекомендуемых документов .....	5
<b>НАСТРОЙКА РАБОТЫ</b> .....	<b>6</b>
Перечень действий при настройке HART .....	6
Алгоритм работы HART-мастера.....	7
<b>ДОБАВЛЕНИЕ И НАСТРОЙКА УСТРОЙСТВ, РАБОТАЮЩИХ ПО ПРОТОКОЛУ HART</b> .....	<b>9</b>
Добавление устройств и объектов в конфигурацию контроллера.....	9
Добавление HART-устройств к модулям аналогового ввода/вывода .....	10
Добавление HART-устройств к коммуникационным модулям .....	12
Добавление HART-устройств непосредственно к модулю центрального процессора.....	14
<b>ДОБАВЛЕНИЕ ПОЛЬЗОВАТЕЛЬСКИХ КОМАНД, ПЕРЕДАВАЕМЫХ ПО HART</b> .....	<b>15</b>
Добавление контейнеров HartDevice .....	15
Использование предустановленного набора команд.....	17
Дизайн собственных команд.....	18
Редактор hart команды .....	18
Библиотека PsIoDrvHartMaster.....	22
<b>ОБЪЯВЛЕНИЕ ПЕРЕМЕННЫХ И ПРИВЯЗКА ПЕРЕМЕННЫХ К КОМАНДАМ</b> .....	<b>24</b>
Объявление переменных .....	24
Привязка переменных программы к командам.....	24
<b>ОБРАБОТКА ВЫПОЛНЕНИЯ КОМАНДЫ</b> .....	<b>28</b>
<b>ПОДДЕРЖКА УСТРОЙСТВ В BURST РЕЖИМЕ</b> .....	<b>30</b>
<b>ФУНКЦИОНАЛЬНЫЙ БЛОК HARTUSERREQUEST И ДИНАМИЧЕСКИ ФОРМИРУЕМАЯ КОМАНДА</b> .....	<b>32</b>
Общее описание .....	32
Объявление структуры данных команды непосредственно в коде .....	33
Использование команды из контейнера команд HART-устройства.....	35
Пример выполнения команды 34 – Write Primary Variable Damping Value .....	36

## ВВЕДЕНИЕ

### Общие сведения

Программное обеспечение контроллера позволяет сконфигурировать его в качестве HART-мастера (по умолчанию – primary) и опрашивать Slave-устройства (интеллектуальные датчики), либо управлять ими (исполнительные устройства с поддержкой HART) по последовательной линии по протоколу HART 6.

Протокол HART основан на методе передачи данных с помощью частотной модуляции (Frequency Shift Keying, FSK), в соответствии с коммуникационным стандартом Bell 202. Цифровая информация передается частотами 1200 Гц (логическая 1) и 2200 Гц (логический 0), которые накладываются на аналоговый токовый сигнал. Скорость передачи данных для HART составляет 1,2 кбит/с.

Работа осуществляется в двух режимах:

- **Одноточечный режим** - соединение «точка-точка», при этом протокол допускает параллельную работу двух мастеров (например, первичный-стационарный контроллер и вторичный- мобильный ручной коммуникатор);
- **Многоточечный режим** - объединение в сеть нескольких ведомых устройств и двух мастеров. При этом по линии осуществляется только цифровая связь. Только одно из ведомых устройств на шине может работать в особом режиме ускоренной передачи (burst mode), в котором оно периодически отправляет в сеть пакет – ответ на заданную команду.

Начиная с версии СПО 1.6.5.0, реализован драйвер на уровне операционной системы (Hart Master OS), с поддержкой протокола HART 7 и технологии FDT/DTM. Подробное описание подключения к контроллеру по спецификации DTM приведено в документе «Настройка и работы REGUL GW DTM. Руководство пользователя».

### Перечень рекомендуемых документов

Для получения информации по настройке других параметров контроллеров серии Regul RX00 в среде разработки Epsilon LD рекомендуется ознакомиться со следующими документами:

- Программное обеспечение Epsilon LD. Руководство пользователя;
- Regul R600. Системное руководство;
- Regul R500. Системное руководство;
- Regul R200. Системное руководство.

## НАСТРОЙКА РАБОТЫ

Установите на компьютер программное обеспечение Epsilon LD. Описание процесса установки программы, а также инструкции по работе с программой приведены в документе «Программное обеспечение Epsilon LD. Руководство пользователя». Программа установки и документация доступны на сайте [www.prosoftsystems.ru](http://www.prosoftsystems.ru).

Запустите программу **Epsilon LD**. Откройте проект, в котором требуется настроить контроллер для обмена данными по протоколу HART. Если такого проекта нет, создайте его с помощью **Мастера конфигурации Regul** (описание приведено в разделе «Основные понятия среды разработки. Проект» документа «Программное обеспечение Epsilon LD. Руководство пользователя DPA-302»).

### Перечень действий при настройке HART

Настройка HART начинается с добавления в проект HART-мастера (**Hart Master**), к которому, в свою очередь, должны быть добавлены конечные устройства **Hart Outer Slave**, непосредственно осуществляющие обмен данными по HART-протоколу. При использовании в проекте модулей ввода/вывода с поддержкой HART (см. соответствующее описание модуля в «Regul RX00. Системное руководство») устройство HART-мастер (**Hart Master**) добавляется непосредственно к модулю ввода/вывода (см. раздел «Добавление HART-устройств к модулю аналогового ввода/вывода») и ассоциируется с выбранным каналом этого модуля, а каждый экземпляр Hart Outer Slave на этом канале обозначает отдельное устройство (датчик) в сети.

В случаях, когда требуется обмен данными с HART-устройствами через внешний HART-модем, имеющий последовательный интерфейс, либо с устройствами, использующими только цифровую часть протокола HART по последовательному каналу, возможны следующие подключения HART-мастера:

- в проект добавляется коммуникационный модуль с нужным количеством последовательных портов (**Extended Regul Serial Port**), к каждому из которых может быть подключен HART-мастер (см. раздел «Добавление HART-устройств к коммуникационным модулям»);
- HART-мастер добавляется непосредственно к последовательному порту (**Regul Serial Port**) модуля центрального процессора (см. раздел «Добавление HART-устройств непосредственно к модулю центрального процессора»).

Далее необходимо создать в проекте объекты типа **HartDevice**. Это контейнеры, содержащие HART-команды для slave-устройства. Следующий шаг – добавление команд в контейнер (см. раздел «Добавление пользовательских команд, передаваемых по HART»).

В МЭК-приложении следует объявить переменные соответствующего типа для каждой команды в контейнере HartDevice. Далее необходимо выполнить привязку контейнера с

командами к slave-устройству и привязку объявленной переменной к параметрам команды (см. раздел «Объявление переменных и привязка переменных к командам»).

Для выполнения HART-команды в МЭК-приложении организуется цикл обработки с анализом статуса команды. И, при переходе команды из фазы выполнения (InProgress) в состояние завершения команды (с ошибкой или без), производится обработка полученных данных. Значение статуса *Ok* устанавливается при успешном получении ответа от устройства и его обработке без ошибок (см. раздел «Обработка выполнения команды»).

### Алгоритм работы HART-мастера

#### Одноточечный режим

Если на один вход/выход аналогового модуля с поддержкой HART подключено одно HART-устройство, то одновременно доступно получение как самого токового значения с датчика (его Primary Value, PV), так и обмен данными с устройством по HART-протоколу. При этом в устройстве, согласно спецификации HART (например, с помощью команды 6), рекомендуется задать адрес 0 и включить режим использования токового сигнала (заводские настройки по умолчанию).

При этом протокол допускает параллельную работу двух мастеров:

- первичное ведущее устройство (Primary);
- вторичное ведущее устройство (Secondary).

#### Многоточечный режим

Если на один вход/выход подключено несколько HART-устройств, то им назначаются уникальные адреса на шине и токовый сигнал переводится в минимально необходимое для функционирования устройства значение (4 мА). При этом работает только цифровая часть HART-протокола – обмен командами. Дополнительно, на уровне **Hart\_Outer\_Slave** осуществляется синхронизация доступа к одному каналу модуля – после захвата и выполнения одной попытки запроса (успешной или с ошибкой) объект **Hart\_Outer\_Slave** гарантированно освобождает канал на заданный период времени (100 мс).



#### ИНФОРМАЦИЯ

Определенные модули ввода/вывода контроллеров серии Regul RX00, оборудованные отдельным HART-модемом на каждый канал, позволяют подключать на выбранный канал до 10 HART-устройств.

В случае, если в модуле один HART-модем с помощью мультиплексирования обслуживает группу каналов, то, для оптимизации обмена данными, рекомендуется подключать на один канал модуля не более одного HART-устройства.

Согласно спецификации HART поддерживается арбитраж шины со вторым мастером и поддержка устройств, работающих в Burst режиме.

### Режим работы каналов модуля

Если каждый канал модуля работает через собственный HART-модем, то выполнение обмена данными по протоколу HART по всем каналам происходит параллельно и независимо от соседнего канала. При этом обработка всех подключенных на один канал устройств с поддержкой HART производится последовательно, по одной команде, с синхронизацией доступа к HART-модему.

Если же каналы в модуле разделены на группы и каждая группа работает через один HART-модем, то производится двухуровневая синхронизация доступа. На первом уровне доступ к HART-модему последовательно пытаются получить экземпляры устройств **Hart Master**, добавленные в дерево устройств к соответствующим каналам модуля в рамках одной подгруппы каналов (например, каналы 1-8 и 9-16 в модуле R500 AI 16 081). Далее, на втором уровне, обработка всех подключенных на один канал устройств с поддержкой HART производится последовательно, по одной команде, также с синхронизацией доступа к HART-модему.



#### **ИНФОРМАЦИЯ**

Рекомендуется производить подключение конечных HART-устройств с равномерным распределением по группам каналов модуля и по каналам модуля в рамках одной подгруппы

---



## ДОБАВЛЕНИЕ И НАСТРОЙКА УСТРОЙСТВ, РАБОТАЮЩИХ ПО ПРОТОКОЛУ HART

### Добавление устройств и объектов в конфигурацию контроллера

Общий принцип добавления устройств/объектов в конфигурацию контроллера описан в разделе «Описание интерфейса. Добавление объектов» документа «Программное обеспечение Epsilon LD. Руководство пользователя». Далее добавляют соответствующие HART-устройства (Рисунок 1)

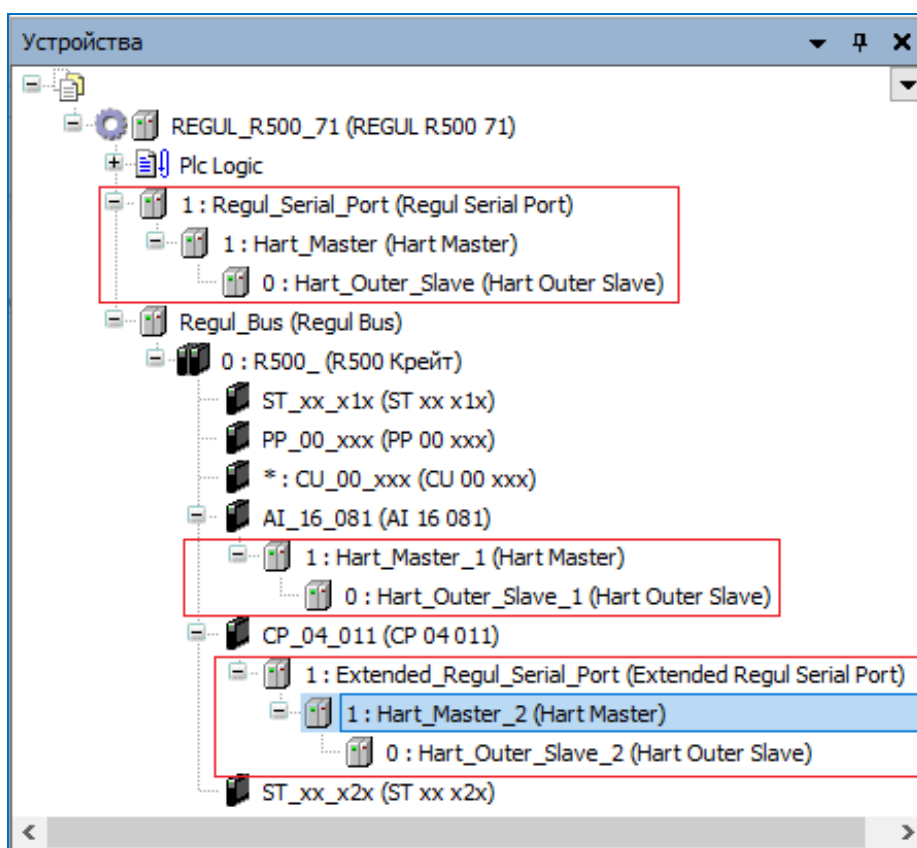


Рисунок 1 – HART-устройства



#### ИНФОРМАЦИЯ

Начиная с версии СПО 1.6.5.0, при выборе устройства Hart доступны варианты драйвера с дополнением OS:

- Hart Master "Prosoft-Systems" Ltd. 1.6.5.0 Устройство, которое работает как Hart мастер на 16 канальных модулях
- Hart Master OS "Prosoft-Systems" Ltd. 1.6.5.0 Устройство, которое работает как Hart мастер на 16 канальных модулях. Драйвер на уровне ОС

Данный драйвер реализован на уровне операционной системы, что позволяет распределить нагрузку от его работы при использовании многоядерных процессоров

## Добавление HART-устройств к модулям аналогового ввода/вывода

Добавьте в проект модуль аналогового ввода/вывода с поддержкой HART. К модулю можно добавить один или несколько HART-мастеров (**Hart Master**), каждый из которых является контейнером для нескольких устройств **Hart Outer Slave**. (Рисунок 2) Максимальное количество HART-мастеров определяется количеством каналов в модуле.

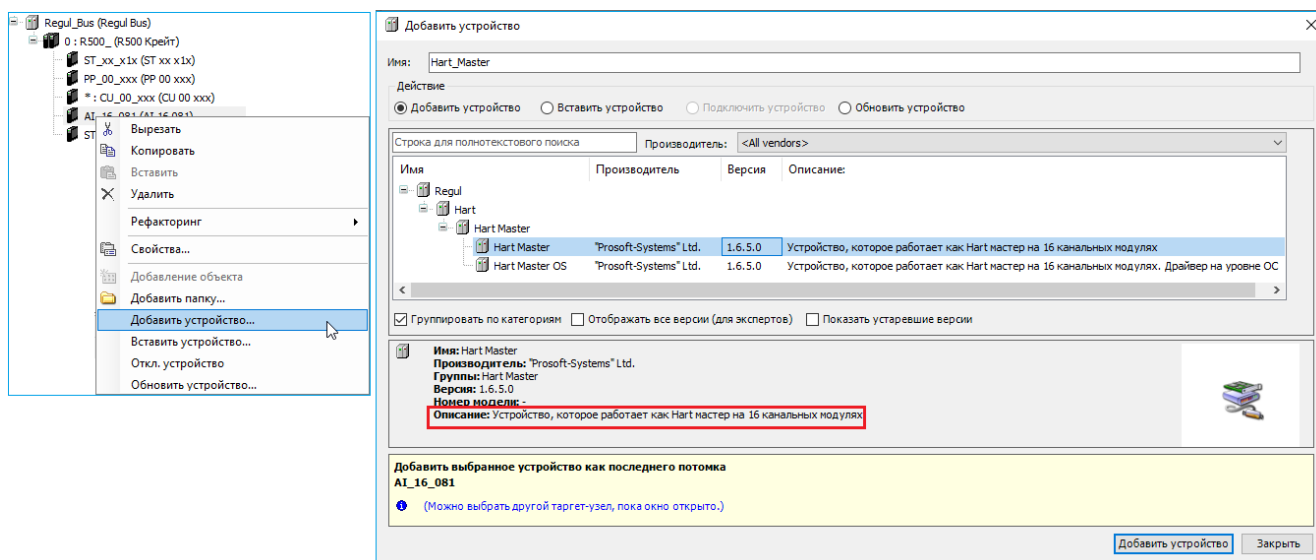


Рисунок 2 – Добавление Hart Master

Далее к устройству **Hart Master** нужно подключить одно или несколько внешних slave-устройств (outer slaves), которые будут опрашиваться контроллером (*Regul* → *Hart* → *Hart Master* → *Hart Outer Slave*) (Рисунок 3).

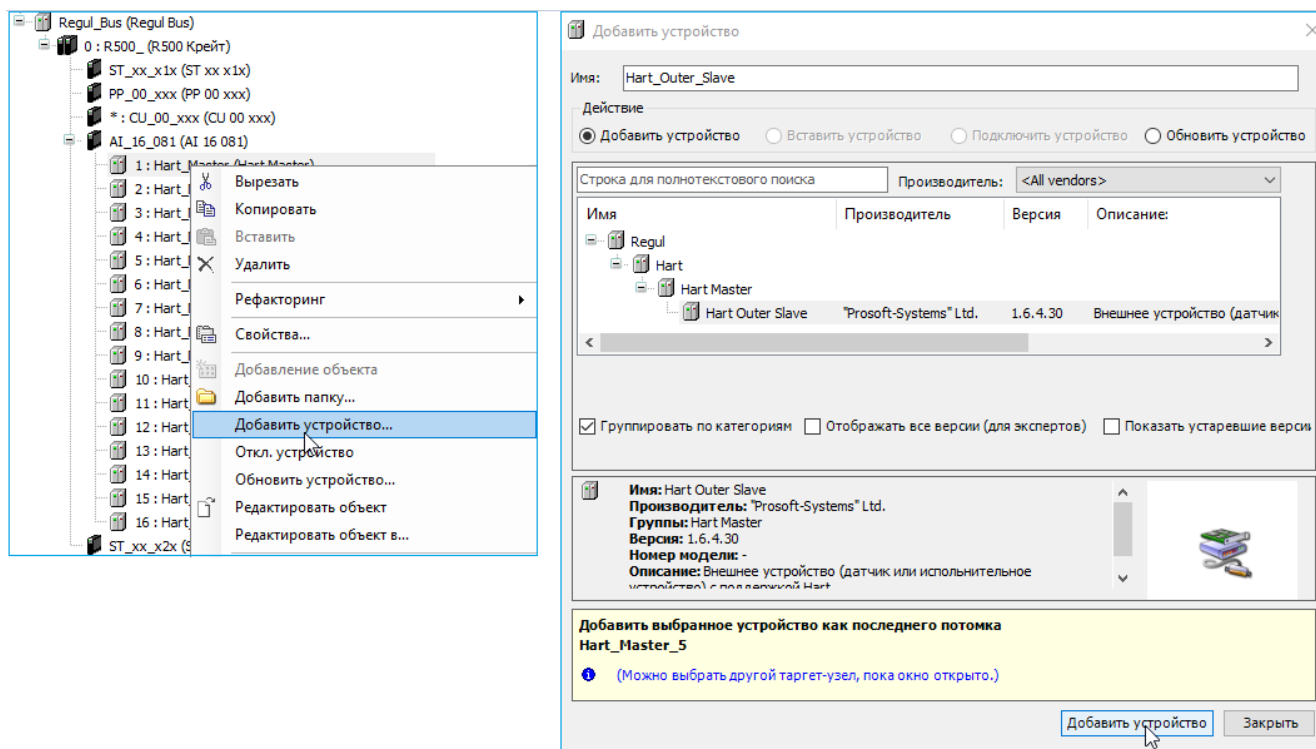


Рисунок 3 – Добавление устройств Hart Outer Slave

Двойным щелчком по названию устройства **Hart Master** откройте вкладку параметров (Рисунок 4).

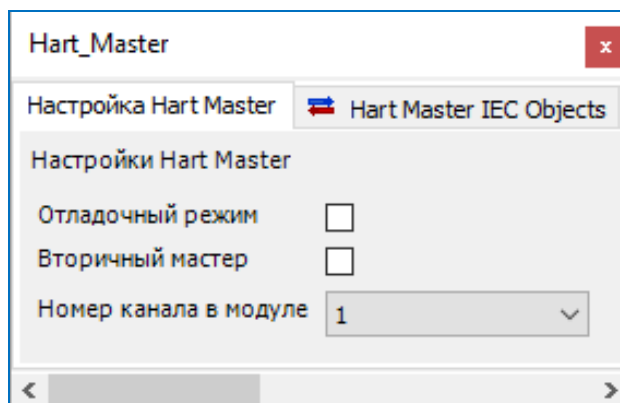


Рисунок 4 – Параметры устройства Hart Master

Для настройки доступны следующие параметры:

- **Отладочный режим** – установка флажка в этом поле включает режим добавления в журнал контроллера отладочных сообщений HART-мастера;
- **Вторичный мастер** – установка флажка в этом поле задает тип мастера как *Вторичное ведущее устройство (Secondary HART Master)*. По умолчанию устройство работает как *Первичное ведущее устройство (Primary HART Master)*;
- **Номер канала в модуле** – номер канала в модуле, к которому подключен и по которому будет опрашиваться устройство Hart Outer Slave.



#### ИНФОРМАЦИЯ

При подключении HART-мастера по последовательному порту не требуется выбор канала в модуле, поле не отображается в редакторе HART-мастера

Двойным щелчком по названию устройства **Hart Outer Slave** откройте вкладку параметров (Рисунок 5).

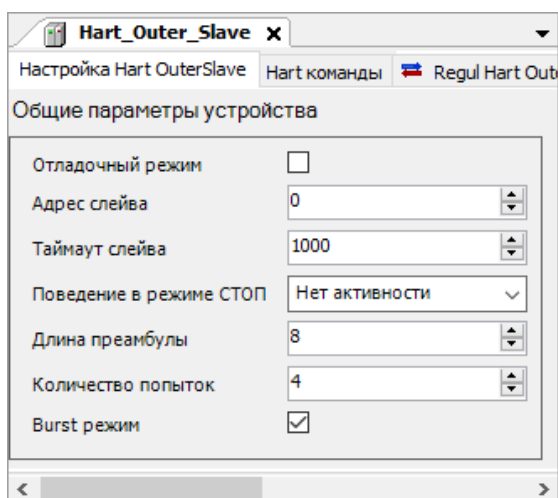



Рисунок 5 – Параметры устройства Hart Outer Slave

Во вкладке **Настройка Hart Outer Slave** доступны следующие параметры:

- **Отладочный режим** – установка флажка в этом поле включает отладочный режим с трассировкой в журнал контроллера;
- **Адрес слейва** – адрес конечного устройства (0-15). Используется в универсальной команде идентификации (Universal Command 0) при начальном опросе устройств на шине;
- **Таймаут слейва** – максимальный таймаут ожидания ответа от HART-устройства, мс;
- **Поведение в режиме СТОП** – указывает, что делать, если переключатель RUN/STOP модуля центрального процессора переведен в положение *STOP*. Возможные значения:
  - *Нет активности* – прекращение опроса,
  - *Нормальная работа* – продолжение работы в обычном режиме,
- **Длина преамбулы** – посылка группы байт 0xFF перед каждой командой, необходимая для синхронизации приемников на шине. Для команды Universal Command 0 всегда используется 20 символов преамбулы;
- **Количество попыток** – число попыток выполнения каждой команды;
- **Burst режим** – установка флажка в поле указывает на присутствие устройства на канале в Burst режиме. При попытке добавить еще одно устройство в Burst режиме, на вкладке **Настройка Hart Outer Slave** высветится знак . Наведя курсор на знак, появится предупреждающая информация: «Burst режим уже установлен для другого устройства».

### Добавление HART-устройств к коммуникационным модулям

Добавьте в крейт коммуникационный модуль RX00 CP XX 011. К нему добавьте объект виртуального последовательного порта **Extended Regul Serial Port**. Предусмотрена возможность добавления нескольких портов расширения.

К каждому порту коммуникационного модуля следует добавить один **Hart Master**. Далее к устройству **Hart Master** нужно подключить одно или несколько внешних slave-устройств (outer slaves), которые будут опрашиваться контроллером (*Regul* → *Hart* → *Hart Master* → *Hart Outer Slave*) (Рисунок 6). Максимальное количество устройств – 10.

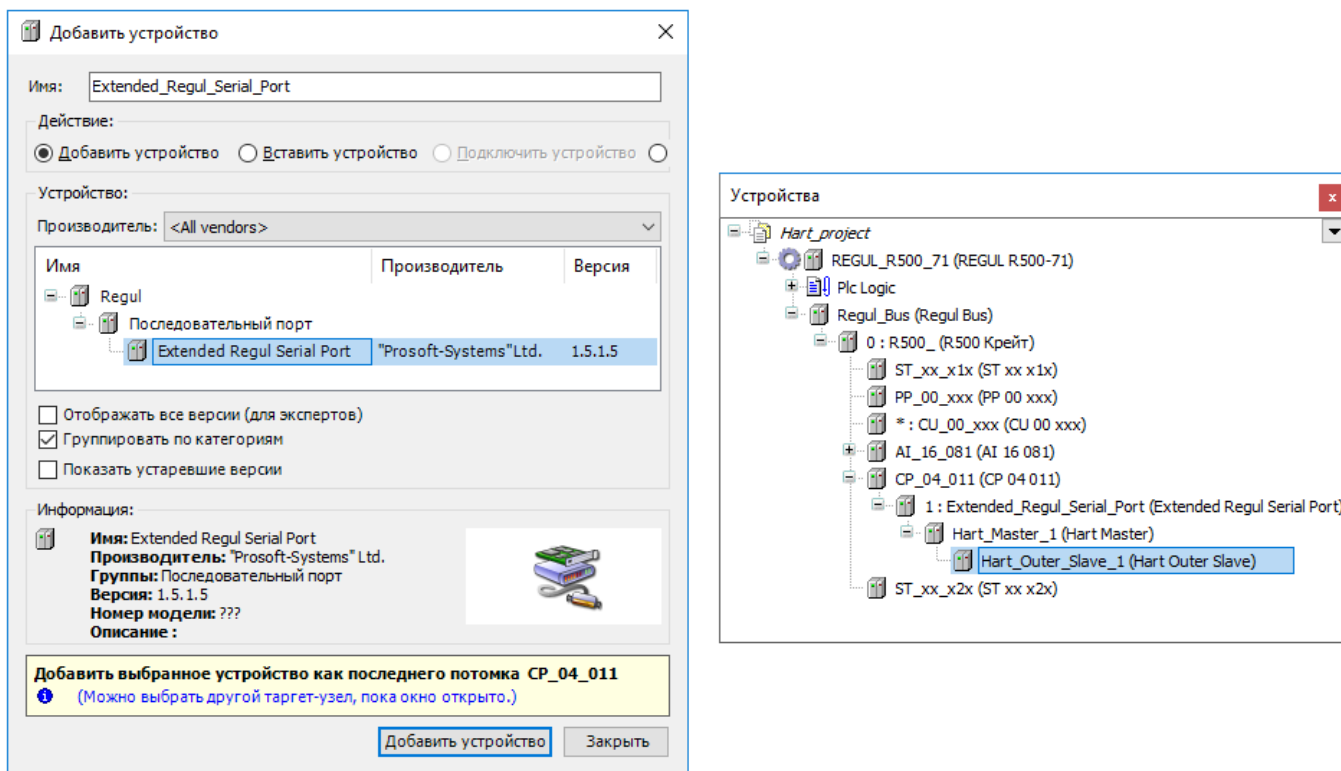


Рисунок 6 – Добавление порта Extended Regul Serial Port и HART-устройств к модулю RX00 CP XX 011

В окне дерева устройств двойным щелчком по названию порта откройте главную вкладку параметров порта. Перейдите на внутреннюю вкладку **Настройки последовательного порта** (Рисунок 7). Номер порта выставляется автоматически (инкрементируется при добавлении нового устройства Extended Regul Serial Port).

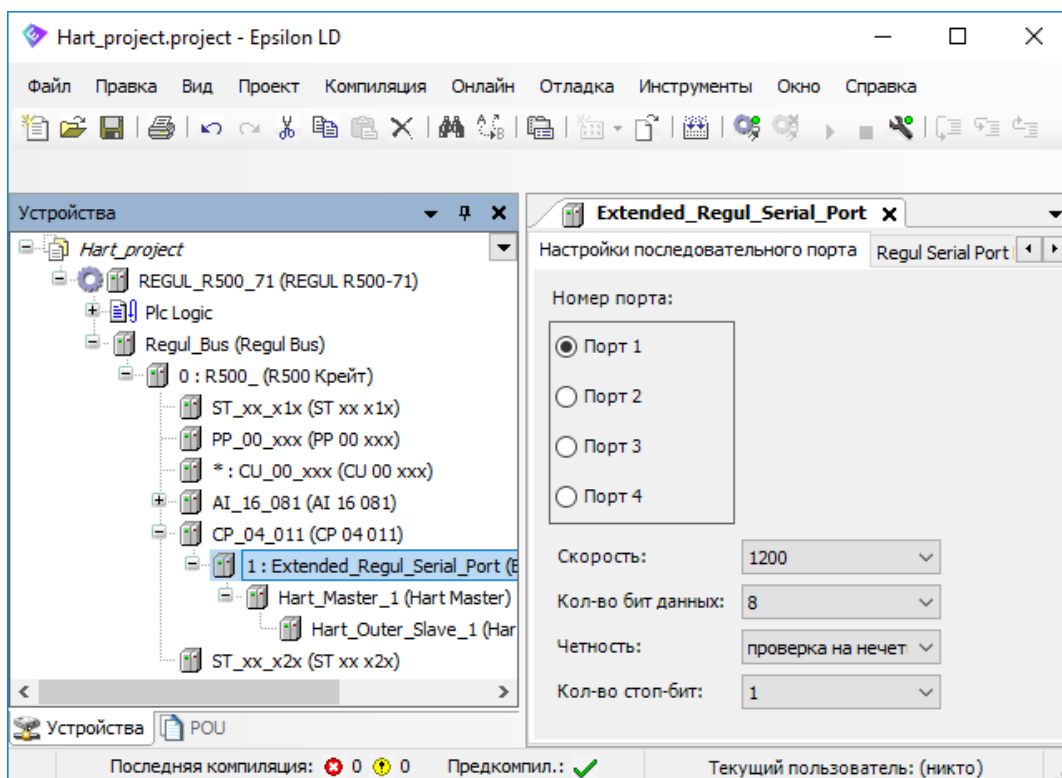


Рисунок 7 – Настройки последовательного порта

Для HART установите следующие параметры:

- **Скорость** – 1200;
- **Кол-во бит данных** – 8;
- **Четность** – проверка на нечетность;
- **Кол-во стоп-бит** – по умолчанию 1.

Настройки устройств Hart Master и Hart Outer Slave аналогичны настройкам, описанным для модулей аналогового ввода/вывода.

## Добавление HART-устройств непосредственно к модулю центрального процессора

Добавьте к головному устройству последовательный порт **Regul Serial Port** (Рисунок 8).

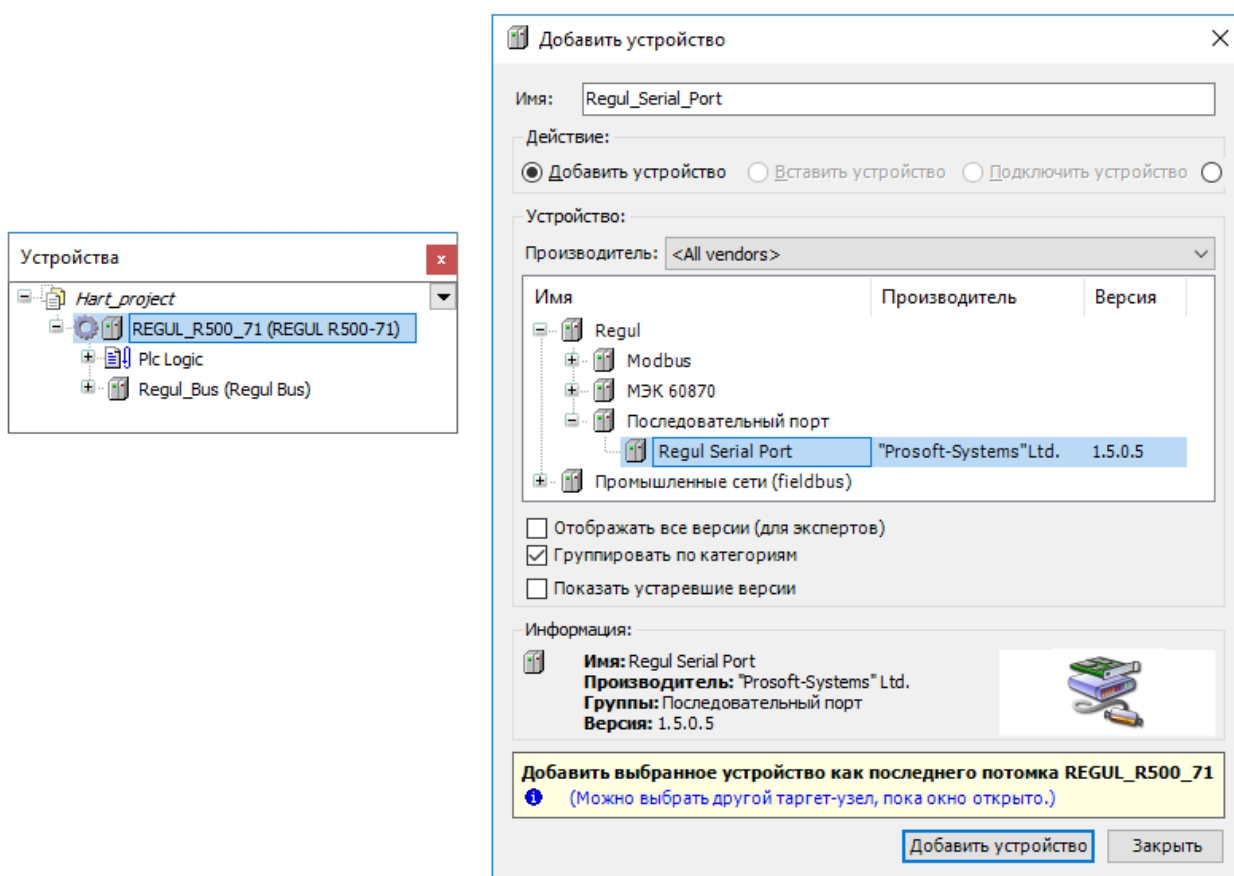


Рисунок 8 – Добавление последовательного порта

Предусмотрено добавление не более двух последовательных портов.

К последовательному порту следует добавить один **Hart Master**. Далее к устройству **Hart Master** нужно подключить одно или несколько внешних slave-устройств (outer slaves), которые будут опрашиваться контроллером (*Regul* → *Hart* → *Hart Master* → *Hart Outer Slave*). Максимальное количество устройств – 10.

Настройки устройств Hart Master и Hart Outer Slave описаны в предыдущих разделах.

## ДОБАВЛЕНИЕ ПОЛЬЗОВАТЕЛЬСКИХ КОМАНД, ПЕРЕДАВАЕМЫХ ПО HART

### Добавление контейнеров HartDevice

Все пользовательские команды, используемые при обмене данными по HART-протоколу, хранятся в специальных объектах **HartDevice**, являющихся контейнерами для команд. Это позволяет группировать команды для конкретных датчиков и других опрашиваемых устройств. Каждый контейнер может содержать множество различных команд. Но к каждому slave-устройству может быть привязан только один контейнер **HartDevice**.

Для добавления контейнера необходимо выполнить следующие действия:

- в окне дерева устройств поставьте курсор на объект **Application**, правой кнопкой мыши вызовите контекстное меню, выберите **Добавить объект... → HartDevice...** (Рисунок 9);

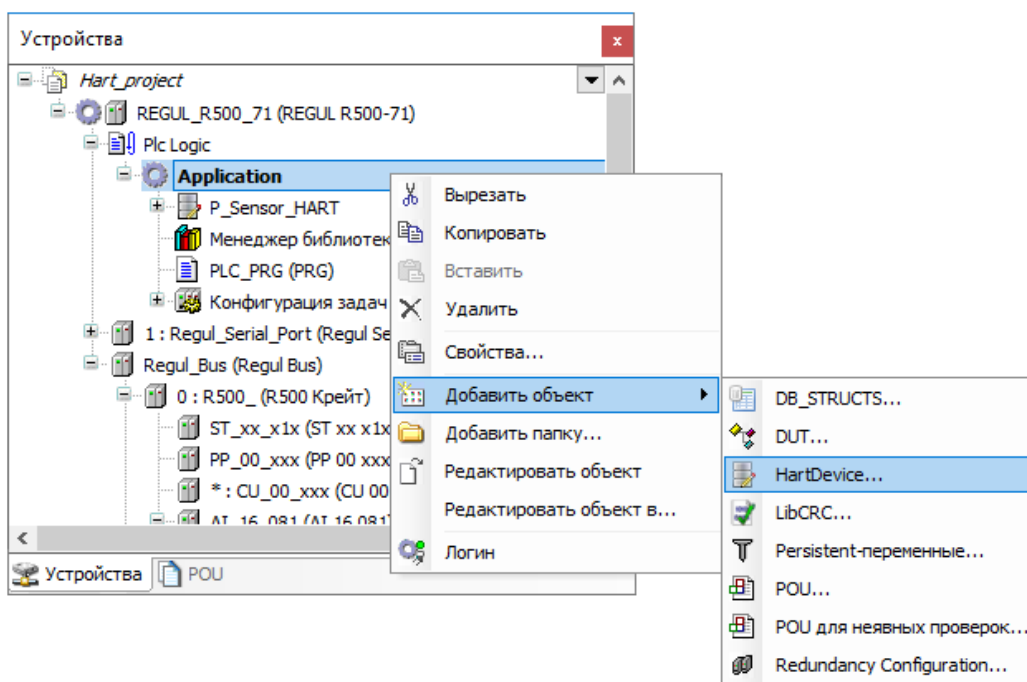


Рисунок 9 – Добавление объекта в конфигурацию контроллера

- откроется окно **Добавить HartDevice**. В поле **Имя** укажите имя контейнера. Например, это может быть название датчика (опрашиваемого устройства) (Рисунок 10). Нажмите кнопку **Добавить**.



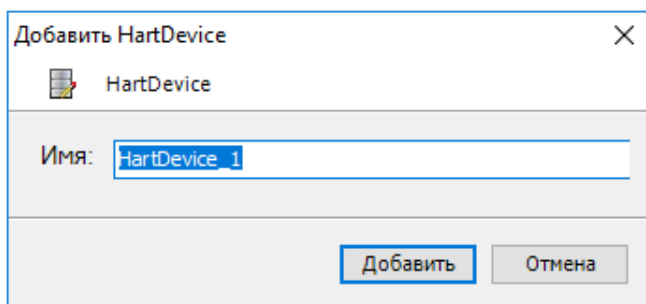


Рисунок 10 – Добавление контейнера



## ИНФОРМАЦИЯ

В дальнейшем при необходимости контейнер можно переименовать (в дереве устройств)

Объект типа HartDevice добавляется в дерево устройств. Автоматически открывается **Редактор hart-устройства** (Рисунок 11).

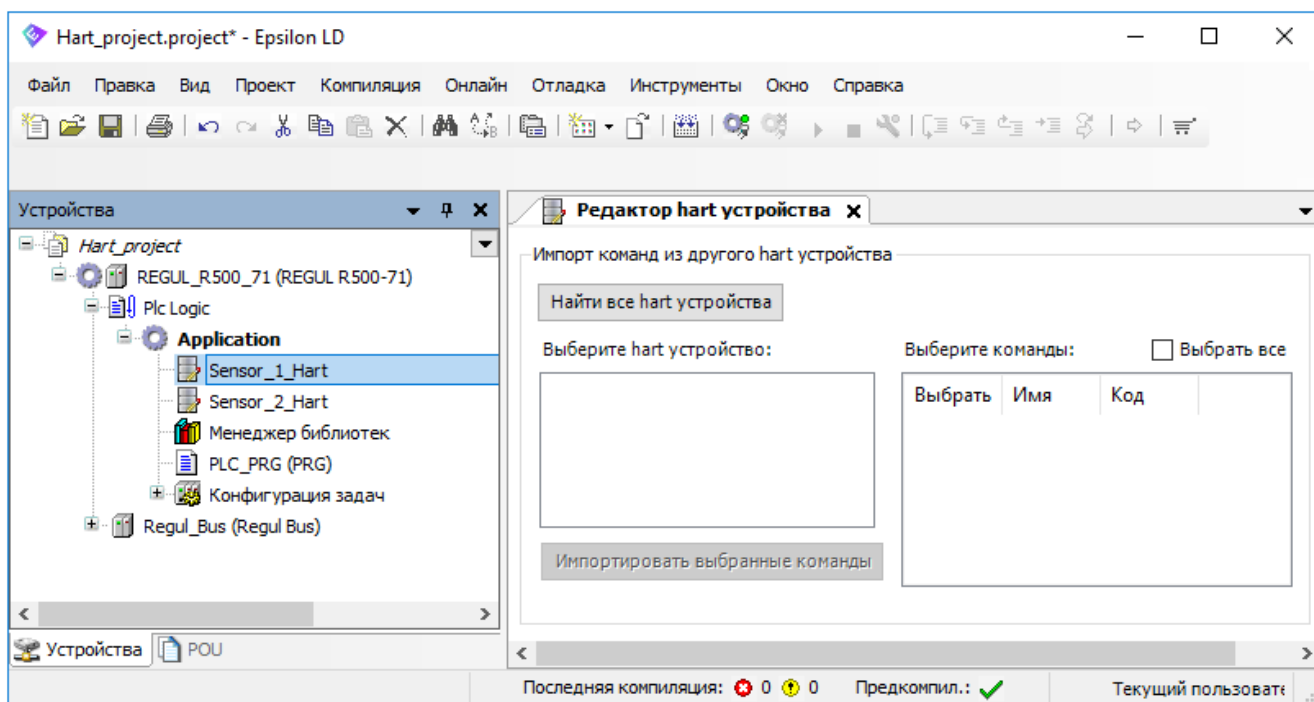


Рисунок 11 – Редактор hart устройства

Нажмите кнопку **Найти все hart устройства**. В поле **Выберите hart устройство:** отобразится список всех контейнеров типа HartDevice, имеющих в текущем проекте и во всех подключенных библиотеках (Рисунок 12).



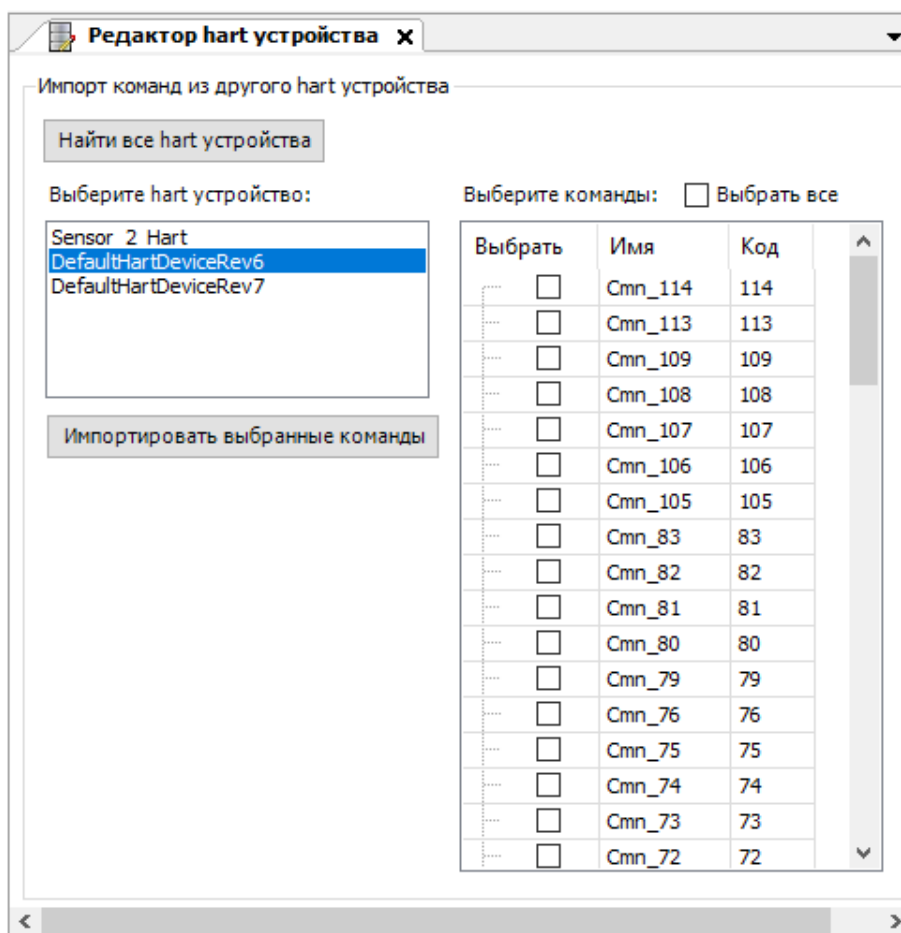


Рисунок 12 – Редактор hart устройства

Объект с именем **DefaultHartDeviceRev6/7** – это контейнер с набором универсальных и общих команд, применяемых в спецификации HART Revision 6.0 и HART Revision 7.0 соответственно (для драйвера, реализованного на уровне операционной системы). Префикс **uni\_** означает, что это команда из спецификации универсальных команд HART (Universal Command Specification), **Cmn\_** – команда из спецификации общепринятых команд (Common Practice Command Specification). Контейнер является частью библиотеки PsIoDrvHartMaster, которая, в свою очередь, автоматически устанавливается вместе с пакетом, содержащим настройки для обмена по HART-протоколу. Таким образом, пользователю по умолчанию доступен контейнер, из которого можно импортировать нужные команды, и не создавать их вручную.

## Использование предустановленного набора команд

Для использования предустановленного набора команд в **Редакторе hart устройства** в поле **Выберите hart устройство:** выберите, например, **DefaultHartDeviceRev6**. В правой части окна в блоке **Выберите команды:** отобразится список всех команд, имеющихся в данном контейнере. Установите флажки для нужных команд и нажмите кнопку **Импортировать выбранные команды**. Команды будут скопированы в пользовательский контейнер (Рисунок 13).

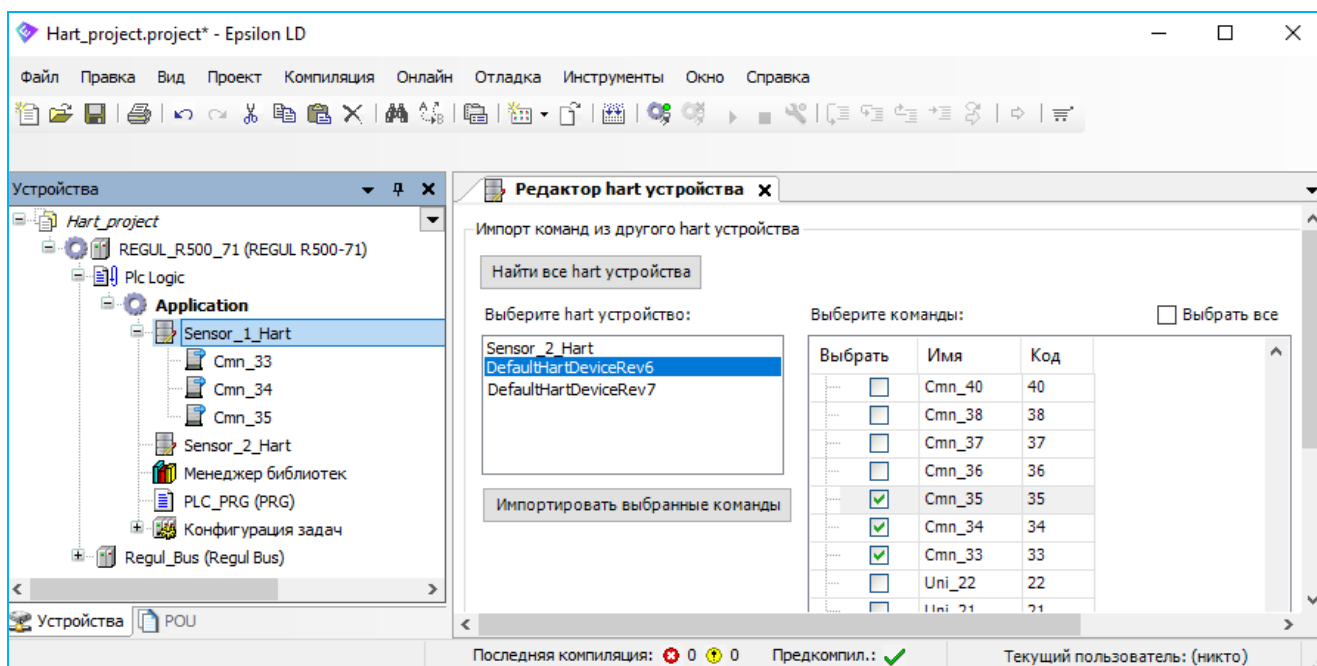


Рисунок 13 – Импорт команд

Для просмотра параметров команды щелкните дважды левой кнопкой мыши по названию команды. Откроется Редактор hart команды (Рисунок 14).

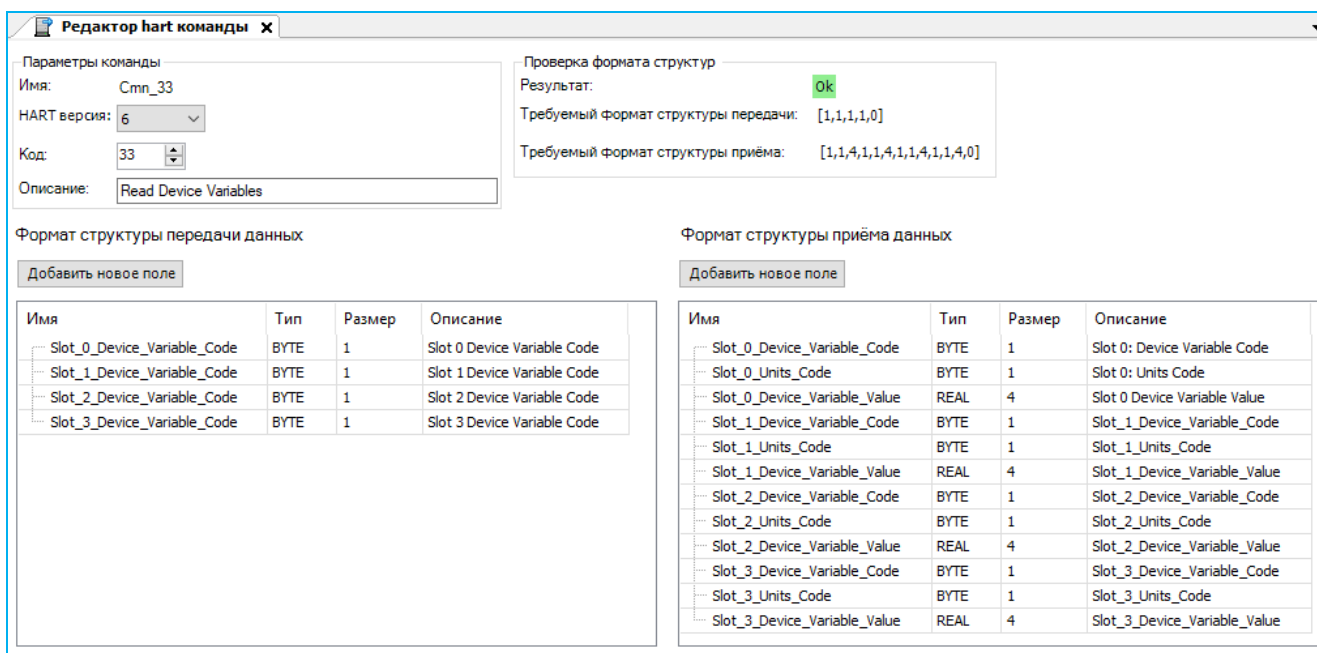


Рисунок 14 – Редактор hart команды

## Дизайн собственных команд

### Редактор hart команды

У пользователя может возникнуть необходимость создания контейнера с новыми командами, отсутствующими в подключенных библиотеках, например, при работе с версиями HART, отличными от ревизии 6.0/7.0. В программе предусмотрена возможность редактирования

существующих команд и создания собственных новых команд. Также можно разработать и предоставить новую библиотеку с контейнером, содержащим требуемые команды.

Для внесения изменений в существующую команду (импортированную или созданную вручную) щелкните дважды левой кнопкой мыши по названию команды. Откроется окно **Редактор hart команды**, где можно изменить код и описание команды, а также добавить/изменить поля в форматах структур приема/передачи данных (Рисунок 15).

При изменении формата структур для определенного диапазона кодов команд (это команды интегрированного в библиотеку HART-устройства DefaultHartDeviceRev6/7) предусмотрена автоматическая проверка формата структур команды по коду. Если формат структур не соответствует спецификации HART для данного кода команды, то в поле **Результат:** появляется сообщение об ошибке (Рисунок 15). При запуске проекта такая команда не будет выполняться.

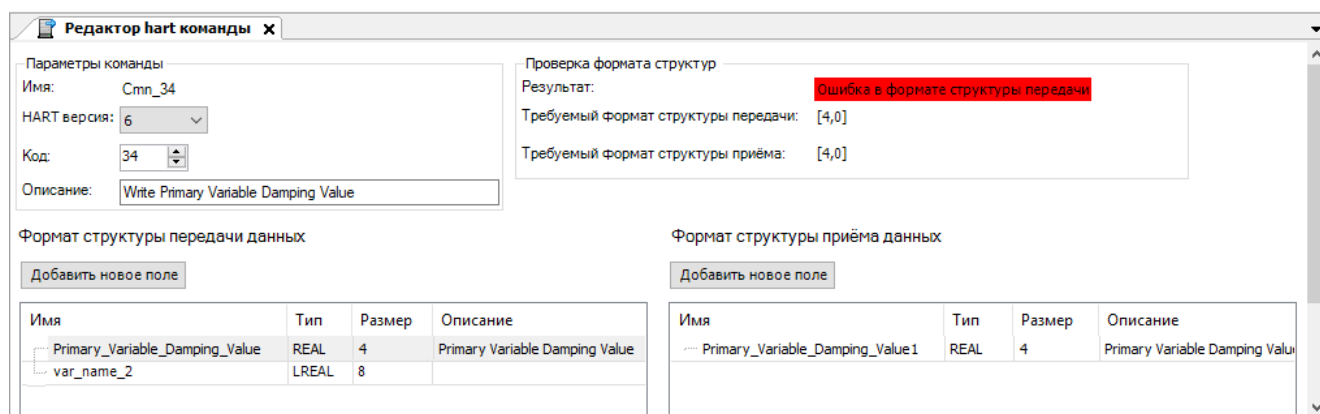


Рисунок 15 – Редактор hart команды

Для создания новой команды поместите курсор на название контейнера. Правой кнопкой мыши вызовите контекстное меню и выберите *Добавить объект* ⇒ *HartCommand...* Откроется окно **Добавить HartCommand**, где в поле **Имя:** задайте имя команды (Рисунок 16).

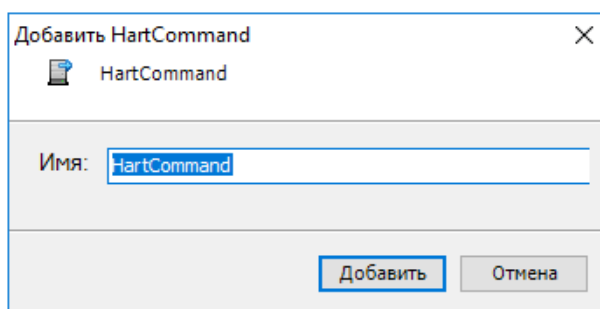


Рисунок 16 – Добавление HART-команды



### ИНФОРМАЦИЯ

В дальнейшем при необходимости команду можно переименовать (в дереве устройств)

Нажмите кнопку *Добавить*. Откроется окно **Редактор hart команды** с пустыми полями (Рисунок 17).

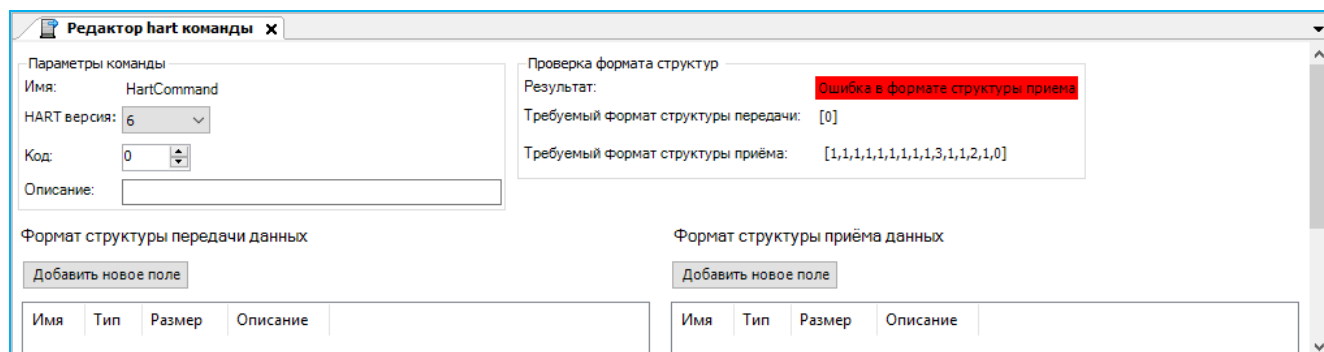


Рисунок 17 – Редактор hart команды при создании новой команды

В поле **HART версия:** выберите значение 6 или 7 из раскрывающегося списка, согласно версии файлов описания и библиотек применяемой спецификации - HART Revision 6.0 или HART Revision 7.0 соответственно.

В поле **Код:** с помощью стрелок или вручную введите код команды – номер команды согласно спецификации HART, соответствующий ее функционалу. Произойдет автоматическая проверка команды по коду. Если команда с таким кодом есть в библиотеке, то в блоке **Проверка формата структур** появляется информация о требуемых форматах структур приема/передачи. Сообщение об ошибке присутствует на этом этапе, т.к. форматы пока не определены.

Если команда не описана в подключенных библиотеках, то в поле **Результат:** появится сообщение: «*Проверка структуры отключена*» (Рисунок 18). Для такой команды нет рекомендаций по форматам структур приема/передачи данных. Такая команда будет добавлена на исполнение.

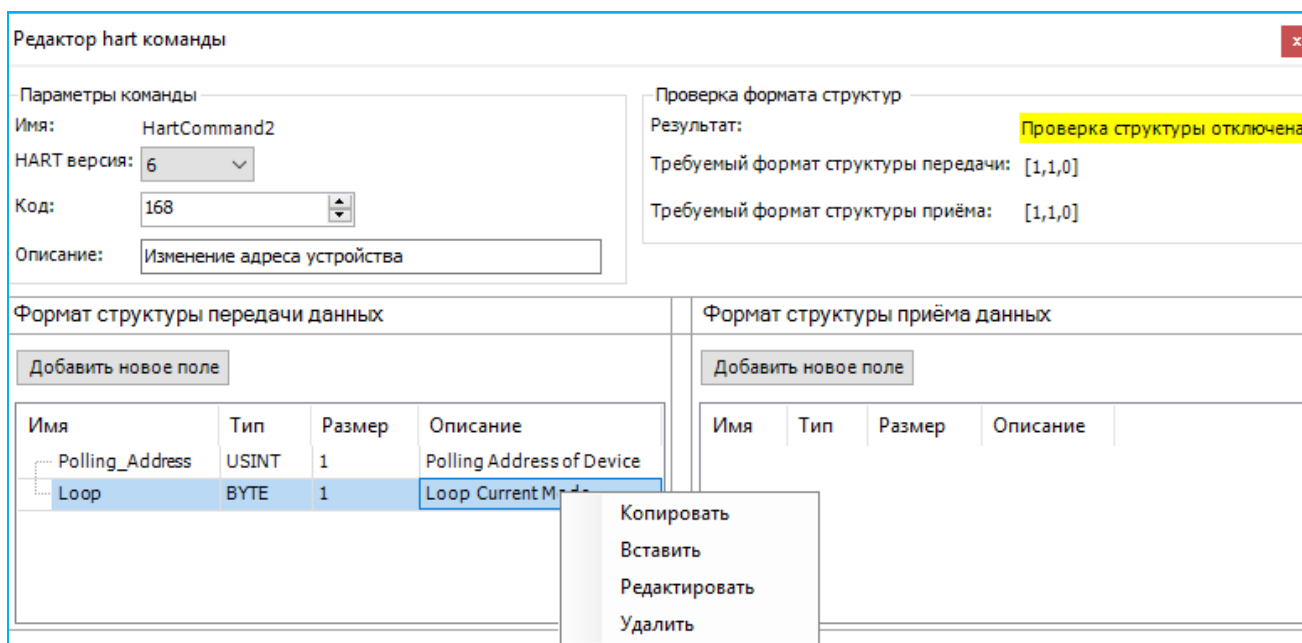


Рисунок 18 – Формирование структур приема/передачи данных

Нажмите кнопку *Добавить новое поле*. Откроется редактор декларации структуры (Рисунок 19).

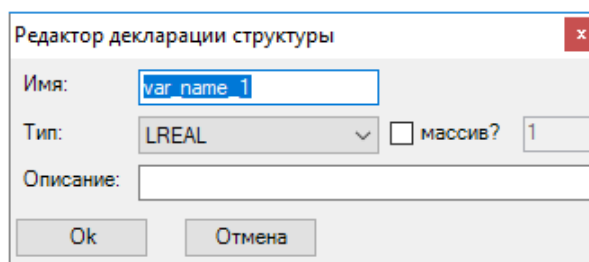


Рисунок 19 – Редактор декларации структуры

Введите имя команды. Выберите тип поля в соответствии со спецификацией HART или вашими требованиями.

Необязательное поле **Описание** заполняется по желанию и несет описательную функцию.

Нажмите кнопку **ОК**. В **Редакторе hart команд** появится созданное поле. В дальнейшем эти поля можно редактировать, удалять, копировать с помощью контекстного меню, вызываемого правой кнопкой мыши.

После того, как все поля добавлены/отредактированы, закройте **Редактор hart команд**. Новая команда с заданными параметрами будет добавлена в контейнер и будет отображаться в дереве устройств.

## Библиотека PsIoDrvHartMaster

В библиотеке **PsIoDrvHartMaster** для каждой команды, указанной в списке пользовательских команд, определены структуры, где поля соответствуют блоку данных как в запросе, так и в ответе. Дополнительно каждая команда содержит четыре поля (Рисунок 20):

- CmdRespCode;
- CmdDevStatus;
- CmdStatus;
- CmdTrigger.

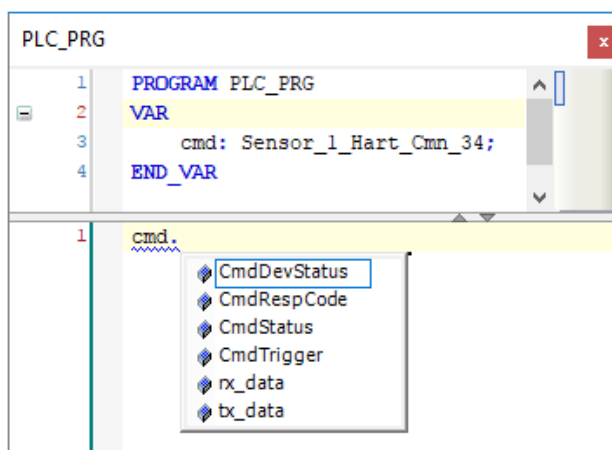


Рисунок 20 – Дополнительные свойства команды

**CmdRespCode** и **CmdDevStatus** – после получения ответа от HART-устройства в эти поля копируются два байта Response Code и Field Device Status, которые, согласно спецификации, содержатся в ответе на любую HART-команду; пользователь может самостоятельно проанализировать их содержимое с точки зрения получения расширенной информации о состоянии опрашиваемого устройства.

**CmdStatus** представляет собой перечисление текущего состояния команды, которое используется в МЭК-приложении для отслеживания цикла обработки команды:

- после добавления команды в очередь она получает статус *Idle* (находится в ожидании начала выполнения по таймеру или по триггеру);
- при переходе в фазу выполнения статус меняется на *InProcess*;
- при успешном получении ответа от устройства и его обработке без ошибок статус устанавливается в значение *Ok*; при этом в поле данных команды **rdata** содержится блок данных ответа от устройства и возможен доступ к его отдельным полям согласно структуре конкретной команды;
- в случае ошибки выполнения команды статус может принимать следующие значения:
  - *Timeout* – от внешнего HART-устройства не был получен полный ответ на команду в течение периода Slave Timeout, заданного для текущего Hart\_Outer\_Slave;

- *ErrorInResponse* – ошибка обработки полученного ответа (неверная длина или неверная структура ответа, неверный адрес, команда не реализована в устройстве, устройство занято, наличие ошибки в байте Response Code);
  - *UserReqArg* – ошибка параметров отдельной динамически формируемой команды, реализованной в функциональном блоке HartUserRequest библиотеки;
  - *InternalError* – ошибка, связанная с обработкой команды внутри библиотеки;
- после выполнения команды и получения ответа ее статус не сбрасывается в *Idle*, а сохраняет свое значение до следующего начала цикла обработки и перехода в состояние *InProcess*;

Полное описание кодов перечисления доступно в МЭК-библиотеке *PsIoDrvHartMaster* (Таблица 1).

Таблица 1 – Коды перечисления команды CmdStatus

Код	Описание
<b>Idle = 1</b>	В ожидании
<b>InProcess = 2</b>	Идет выполнение цикла "запрос-ответ"
<b>Ok = 3</b>	Данные достоверны, команда успешно выполнена
<b>Timeout = 4</b>	Ошибка - нет ответа
<b>ErrorInResponse = 5</b>	Ошибка - ошибка в ответе
<b>UserReqArg = 6</b>	Ошибка - некорректный аргумент в пользовательском запросе
<b>InternalError = 7</b>	Ошибка - внутренняя ошибка обработки команды

**CmdTrigger** – управляющий флаг для команд по требованию (Trigger); в библиотеке фиксируются все переходы триггера из состояния *False* (0) в состояние *True* (1) и команда будет добавлена в очередь выполнения заданное число раз.

## ОБЪЯВЛЕНИЕ ПЕРЕМЕННЫХ И ПРИВЯЗКА ПЕРЕМЕННЫХ К КОМАНДАМ

### Объявление переменных

Для того, чтобы HART-команды выполнялись, требуется добавить в программном коде переменные, представляющие из себя структуры определенных типов, описанных в библиотеке PsIoDrvHartMaster (PsIoDrvHartMaster\_OS), и соответствующие добавленным в пункте выше командам для устройства Hart Outer Slave. Для всех добавленных HART-команд автоматически генерируются новые типы структур с именами, состоящими последовательно из имени контейнера (HartDevice), затем символа подчеркивания и имени переменной (HartCommand). Например, для контейнера с именем **Metran\_P\_Sensor** и командой **Read\_Primary\_Variable** будет сгенерирован тип структуры с именем *Metran\_P\_Sensor\_Read\_Primary\_Variable*.

Для создания переменных откройте редактор ПЛК-программы (МЭК-приложение). Например, в редакторе ST для программы MAIN создание переменной выглядит следующим образом: *cmd: название контейнера\_название команды*. (Рисунок 21).

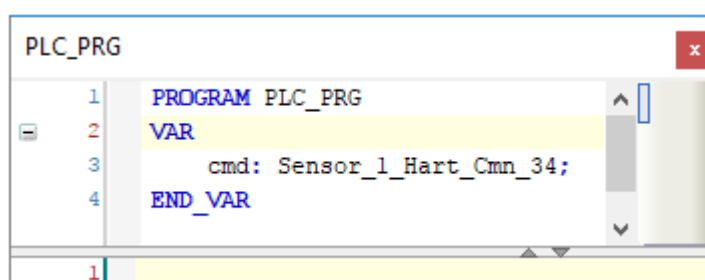


Рисунок 21 – Пример объявления переменной

### Привязка переменных программы к командам

Первым шагом следует привязать контейнер к slave-устройству, далее привязать переменные программы к командам. К одному slave-устройству допустимо привязать только один контейнер с командами.

Двойным щелчком по названию устройства **Hart Outer Slave** откройте редактор устройства. Перейдите на вкладку **Hart Команды** (Рисунок 22).



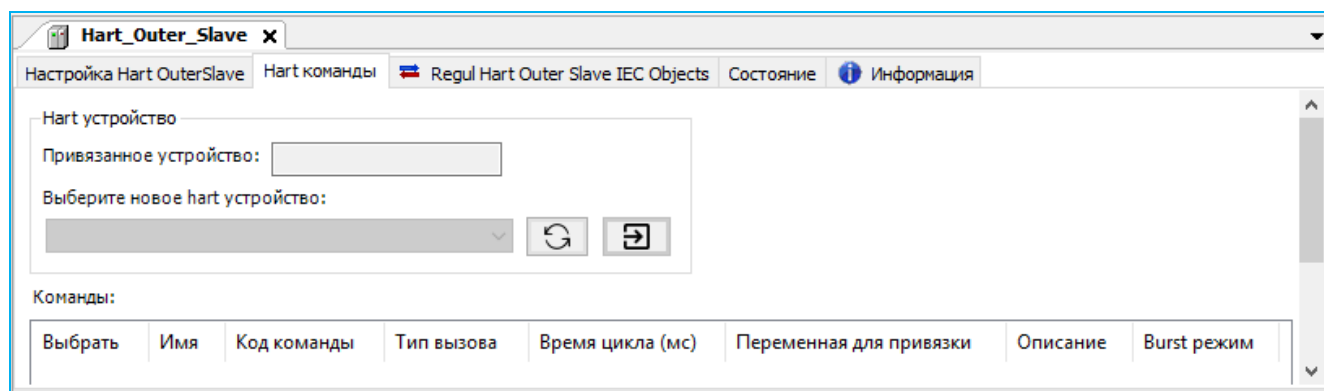




Рисунок 22 – Вкладка «Hart Команды»

Нажмите кнопку поиска устройства . В поле **Выберите новое hart устройство:** найдите в раскрывающемся списке название контейнера, выберите его. Нажмите кнопку . В поле **Привязанное устройство:** появится название контейнера, а в поле **Команды:** – список команд, которые содержит этот контейнер (Рисунок 23).

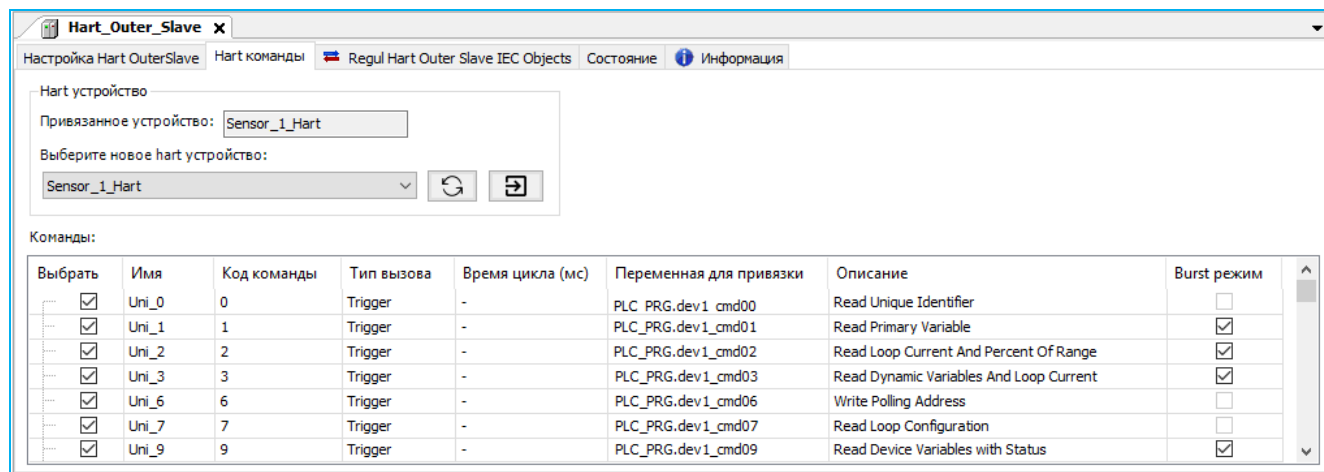



Рисунок 23 – Контейнер привязан к slave-устройству

В строке нужной команды установите флажок в поле **Выбрать** и станут доступны для редактирования параметры команды.

В поле **Тип вызова команды** выбирается, как команда будет добавляться в очередь выполнения, при этом доступны следующие значения для настройки:

- *заданный период (Timer)* – период выполнения команды, мс. Для редактирования будет доступно поле **Время цикла**,
- *по требованию (Trigger)* – для добавления команды в очередь выполнения. Во время выполнения МЭК-приложения каждый раз, когда значение этого поля для переменной, соответствующей добавленной команде, переходит из False в True команда добавляется в очередь,

В поле **Переменная для привязки** можно указать необходимую переменную. Имя переменной можно ввести вручную или через кнопку , открывающую окно **Ассистент ввода**. Раскрывая иерархический список, найдите нужную переменную (Рисунок 24).

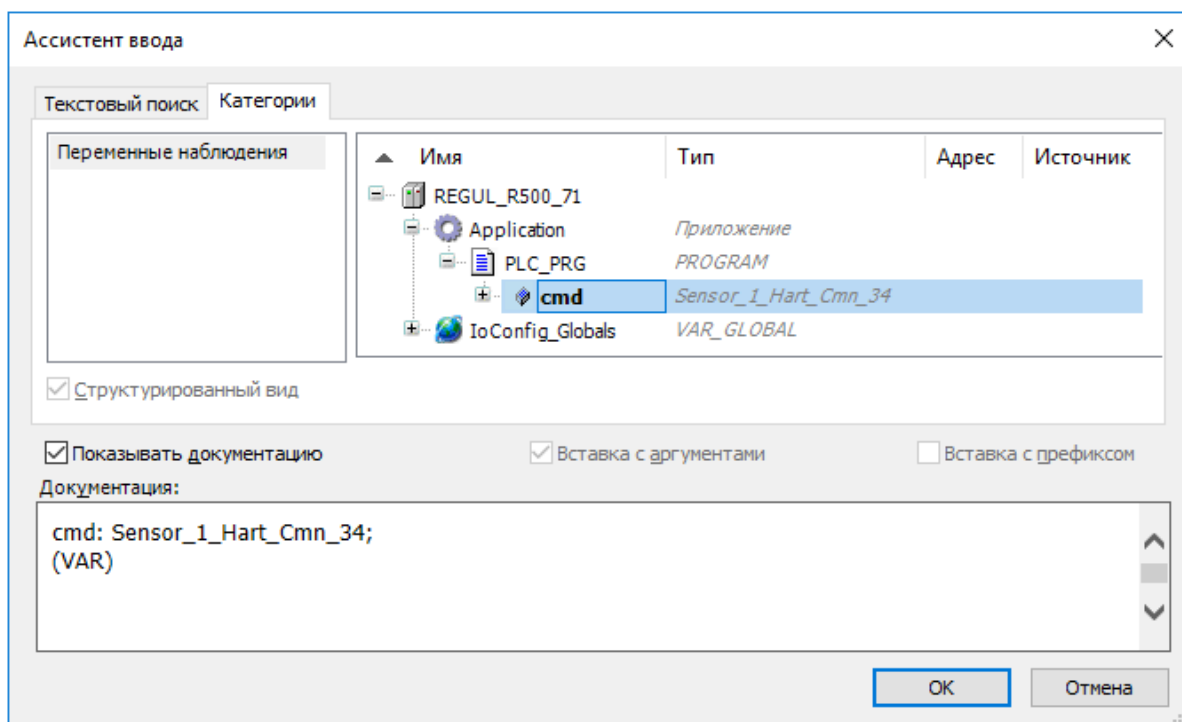


Рисунок 24 – Ассистент ввода

Выберите переменную для привязки. Окно **Ассистент ввода** закроется, а в строке команды в поле **Переменная для привязки** появится название переменной (Рисунок 25). Для сохранения поместите курсор в любое место окна.

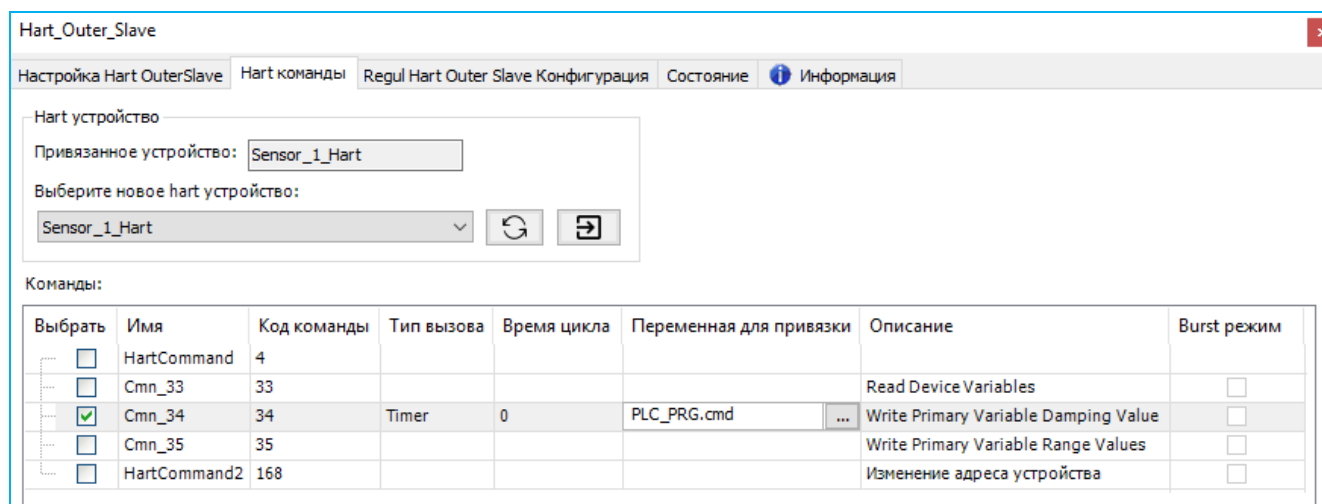


Рисунок 25 – Переменная для привязки

В поле **Burst режим** пользователь может установить флажок, указывающий, что данная команда будет использоваться для обработки Burst-пакетов, полученных мастером в сети от соответствующих устройств (см. раздел «Поддержка устройств в Burst режиме»).

К каждой команде можно привязать только переменную соответствующего типа. В случае несовпадения типов появится сообщение об ошибке (Рисунок 26).

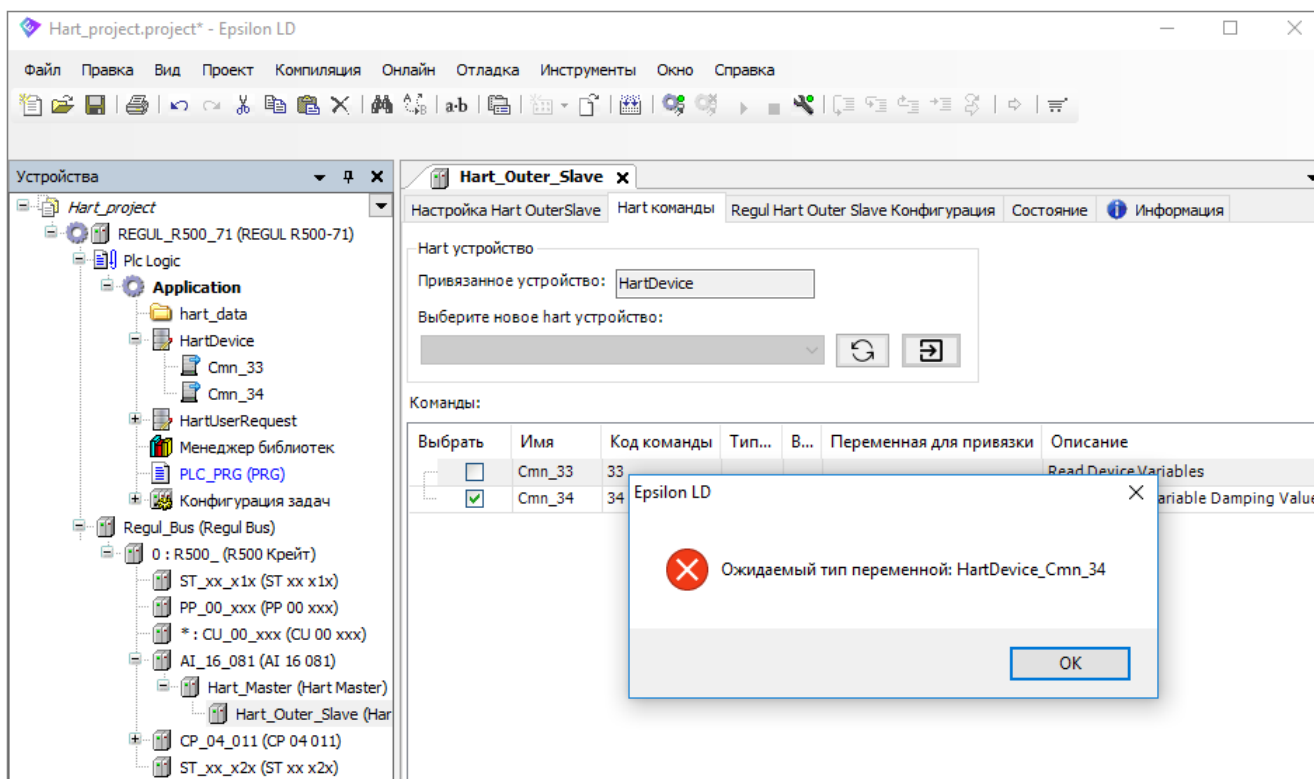


Рисунок 26 – Сообщение об ошибке при несовпадении типов переменных

## ОБРАБОТКА ВЫПОЛНЕНИЯ КОМАНДЫ

Обработка выполнения команды **по таймеру** состоит из следующих шагов:

- проверьте поле статуса выполнения команды. Если статус равен `CmdStatus.Ok`, то команда была выполнена успешно и можно работать с полученными данными;
- ошибка в статусе команды означает сбой при выполнении команды либо наличие ошибки в самом ответе от устройства и требуется дополнительный анализ кода `CmdStatus` либо анализ полей команды `CmdRespCode` и `CmdDevStatus`.

Обработка выполнения команды **по триггеру** состоит из следующих шагов:

- для добавления команды в очередь выполнения по триггеру установите в коде программы для поля `CmdTrigger` команды значение `True` (предварительно это поле должно быть выставлено в `False`):

```
IF Условие_запуска_команды_на_выполнение = TRUE THEN
    P_Sensor_cmd01.CmdTrigger := TRUE;
END_IF
```

- отслеживайте успешное выполнение команды по полю статуса:
  - если статус равен `CmdStatus.Ok`, то команда в последнем цикле была выполнена успешно и можно работать с полученными данными;
  - ошибка в статусе команды означает сбой при выполнении команды либо наличие ошибки в самом ответе от устройства и требуется дополнительный анализ кода `CmdStatus` либо анализ полей команды `CmdRespCode` и `CmdDevStatus`.

Обработка полученных значений в поле данных команды:

```
IF P_Sensor_cmd01.CmdStatus = CmdStatus.Ok THEN
    byteValue1 := P_Sensor_cmd01.rx_data.Primary_Variable_Units;
    realValue2 := P_Sensor_cmd01.rx_data.Primary_Variable;
END_IF
```

Обработка выполнения **Burst** команды состоит из следующей последовательности действий:

- проверьте поле статуса выполнения команды. Если статус равен **`CmdStatus.Ok`**, то **Burst** пакет был обработан успешно и можно работать с полученными данными;
- ошибка в статусе команды **`CmdStatus.ErrorInResponse`** означает сбой при обработке **Burst** пакета, либо наличие ошибки в самом ответе от устройства, и требуется дополнительный анализ кода **`CmdStatus`**, либо анализ полей команды **`CmdRespCode`** и **`CmdDevStatus`**.



### **ИНФОРМАЦИЯ**

При выполнении пользовательских команд используется «длинный» (5 байт) уникальный адрес устройства, который формируется в библиотеке при получении ответа на запрос команды идентификации (Universal Command #0). Поэтому, если уникальный адрес еще не сформирован, то перед выполнением пользовательской команды библиотека автоматически добавляет команду идентификации и пытается ее выполнить. Также эта команда будет автоматически выполняться в случае ошибки обмена с устройством перед следующей в очереди пользовательской командой


## ПОДДЕРЖКА УСТРОЙСТВ В BURST РЕЖИМЕ

**Burst** режим позволяет устройству циклически отправлять в сеть оперативные данные без запроса со стороны мастера. Только одно устройство в сети может быть переведено в этот режим. Для настройки Burst режима используются команды, описанные в таблице 2:

Таблица 2 – Команды для настройки Burst режима

Команда	Описание
<b>Command 109</b> Burst Mode Control	Включение и выключение Burst режима
<b>Command 108</b> Write Burst Mode Command Number	Задаёт выбор номера команды, ответ на которую устройство будет транслировать в сеть. К таким командам относятся команды 1, 2, 3, 9 и 33. Команда может выполняться как на этапе конфигурации устройства, так и в процессе работы в режиме Burst «на лету»
<b>Command 107</b> Write Burst Device Variables	Позволяет выбрать номера Device Variables в устройстве, которые будут переданы в ответе на команду 9 и 33
<b>Command 105</b> Read Burst Mode Configuration	Чтение текущей конфигурации Burst

Для организации получения данных от устройства в Burst режиме, подключенном на один из каналов, необходимо выполнить следующие действия:

1. Добавьте HART-устройство на канал (**HART Master**⇒**Hart Outer Slave**). Установите в общих параметрах устройства флажок в поле **Burst режим**.
2. Добавьте контейнер **HartDevice** (**Application**⇒**Добавить объект...**⇒**HartDevice...**). На вкладке **Редактор hart устройства** выберите hart устройство и импортируйте в контейнер необходимые для настройки (105,107,108,109) и получения данных (1,2,3,9,33) команды.
3. Перейдите на вкладку **Hart команды** объекта **Hart Outer Slave** (п.1). Нажмите кнопку . Активируйте созданный в п.2 контейнер команд, установив флажок в поле **Выбрать**. Далее установите флажок в поле **Burst режим** для тех команд, по которым будут приходиться оперативные данные от устройства в Burst режиме.



### ВНИМАНИЕ!

В контейнер команд **HartDevice** можно добавить только один экземпляр команды каждого конкретного типа. При этом команда может обрабатываться либо в обычном режиме (запрос/ответ, циклически или по триггеру), либо в Burst режиме (получение пакетов от Burst устройства в режиме прослушивания сети, только команды 1, 2, 3, 9, 33), в зависимости от состояния флажка команды из п.3. Тип вызова команд не важен, если выставлен флажок в поле Burst режим

- Добавьте в код переменные соответствующих типов (PLC\_PRG). В экземпляры этих переменных будут копироваться данные из полученных Burst пакетов. В списке команд из п.3 на вкладке **Hart команды** привяжите добавленные переменные к активным командам с помощью ассистента ввода в поле **Переменная для привязки**. (Рисунок 27).

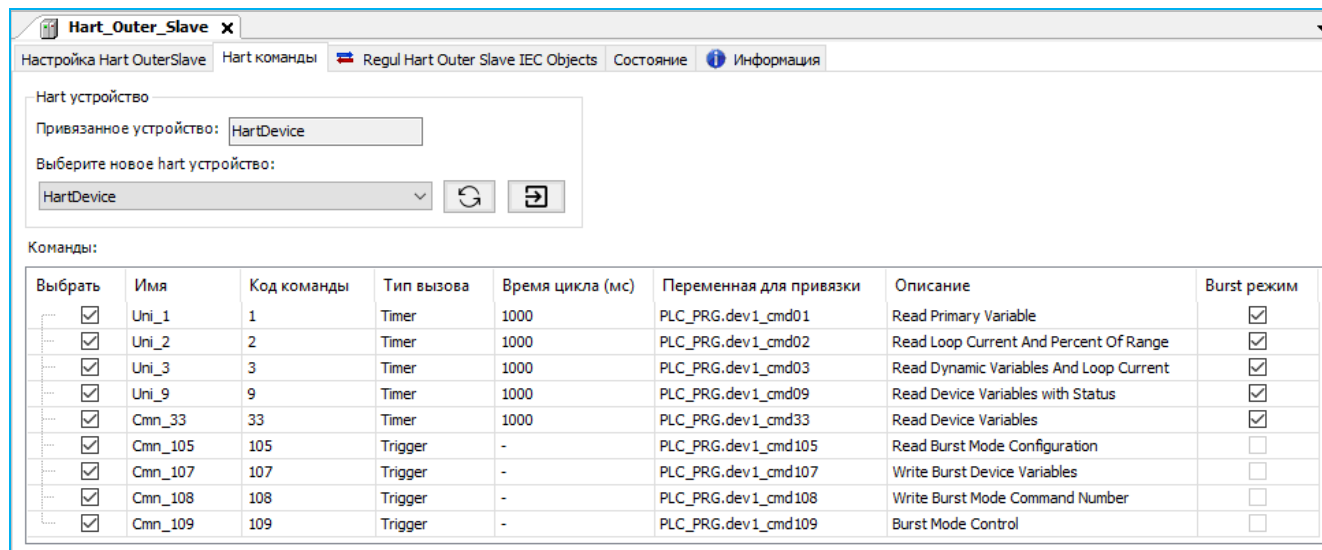


Рисунок 27 - Добавление и привязка команд для Burst режима

- После загрузки и запуска приложения на контроллере экземпляр объекта **Hart Outer Slave** переходит в рабочий режим – выполнение обычных команд (запрос/ответ) и прослушивание сети на предмет обнаружения Burst пакетов. Уникальность команд в контейнере **HartDevice**, привязанном к объекту **Hart Outer Slave**, позволяет однозначно сопоставить экземпляр переменной МЭК-приложения и данные в полученном Burst пакете. В случае успешной обработки пакета, данные из него, включая байты **Response Code** и **Field Device Status**, копируются в соответствующие поля структуры переменной МЭК-приложения, а поле **CmdStatus** принимает значение *Ok* (3). В случае ошибки поле **CmdStatus** устанавливается в значение *ErrorInResponse* (5).

## ФУНКЦИОНАЛЬНЫЙ БЛОК HARTUSERREQUEST И ДИНАМИЧЕСКИ ФОРМИРУЕМАЯ КОМАНДА

### Общее описание

В библиотеке имеется возможность работы в МЭК-приложении с динамически формируемой командой HART, реализованной в функциональном блоке (далее – ФБ) *HartUserRequest*. Работа с командой – только по триггеру. Burst команды не поддерживаются.

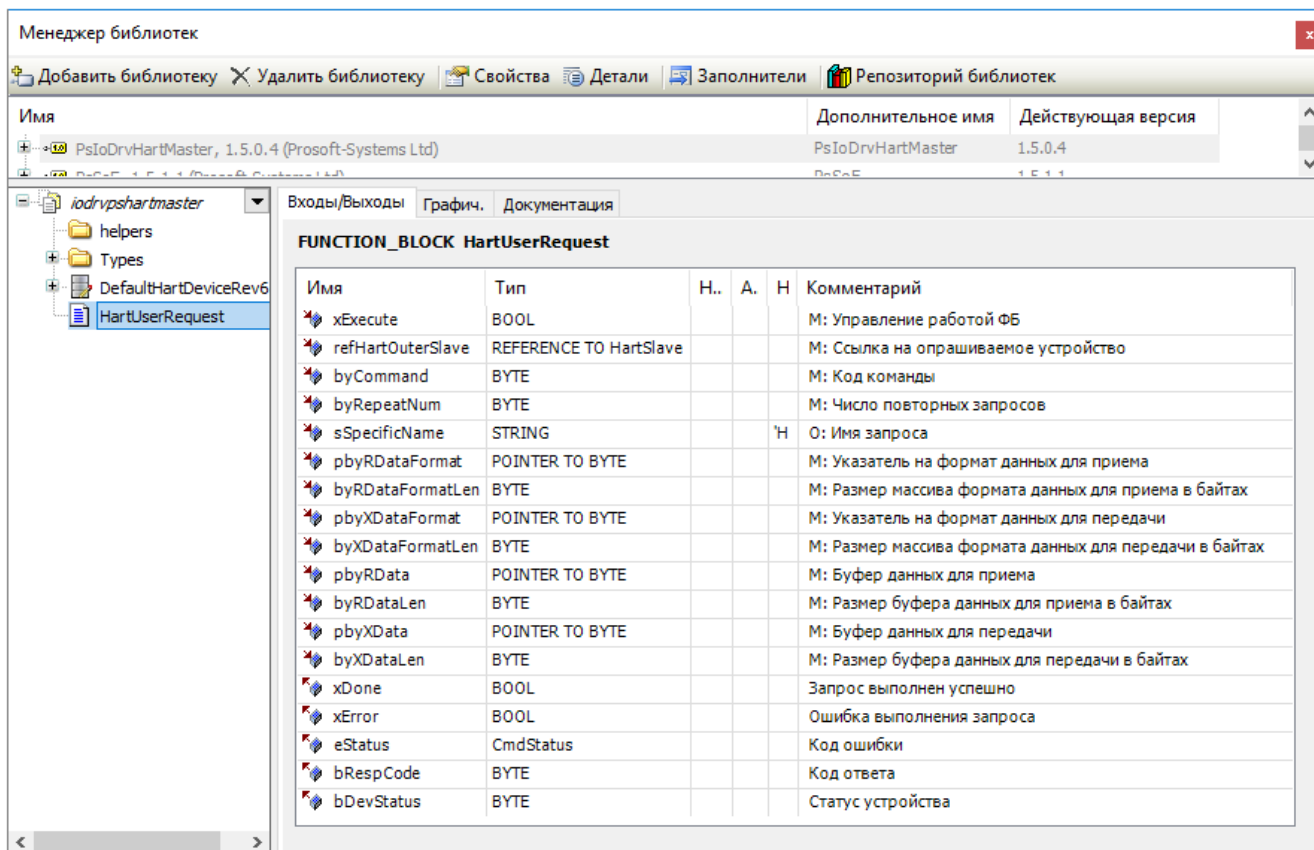


Рисунок 28 – Функциональный блок HartUserRequest

Для корректной обработки произвольной HART-команды в переменную ФБ *HartUserRequest* при ее вызове в МЭК-приложении передаются специфические параметры, включающие форматы структур поля данных для записи (в запросе) и чтения (в ответе). Это позволяет библиотеке корректно отформатировать выходной/входной блок данных запроса/ответа и сопоставить разнотипные параметры, содержащиеся в этом блоке, соответствующим полям переменной команды, объявленной в МЭК-приложении.

Ниже приведен пример выполнения универсальной команды запроса информации об устройстве (команда с кодом 15). При этом доступны два варианта задания структуры данных.



## Объявление структуры данных команды непосредственно в коде

Для объявления структуры данных команды непосредственно в коде необходимо наследовать структуру 15-й команды от базовой структуры CmdBase, объявленной в библиотеке (Рисунок 29).

```
//UNI 15 - Read Device Information;
TYPE cmd15 EXTENDS PsIoDrvHartMaster.CmdBase :
STRUCT
    data : cmd15_rdata;    //UNI 15 - Read Device Information;
END_STRUCT
END_TYPE
```

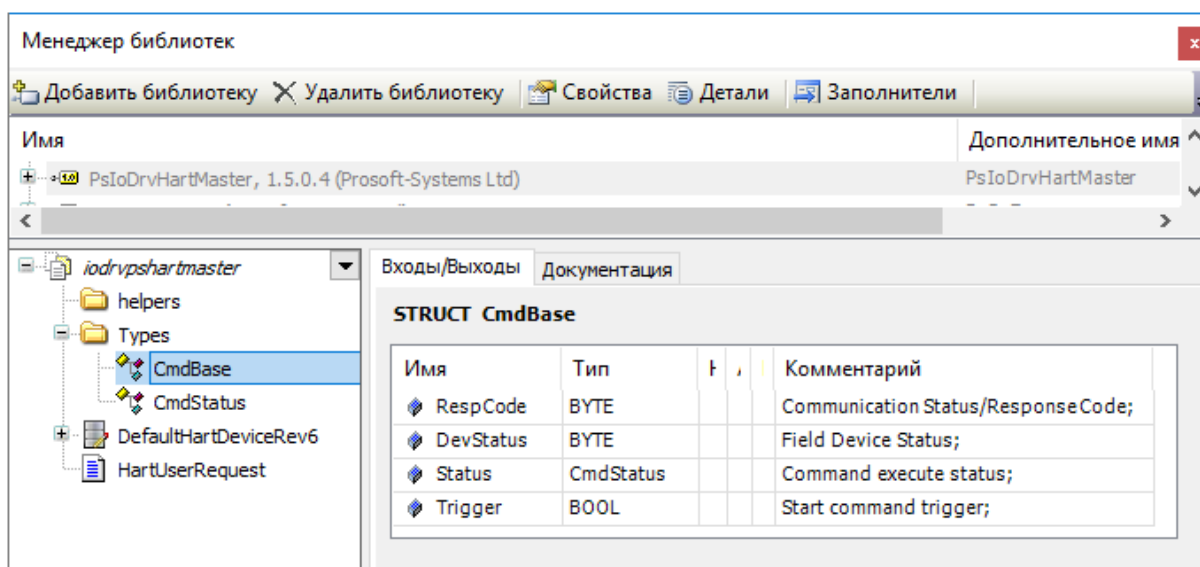


Рисунок 29 – Базовая структура CmdBase

С полем данных типа структуры cmd15\_rdata, соответствующей спецификации HART.

```
{attribute 'pack_mode' := '1'}
TYPE cmd15_rdata :    //UNI 15 - Read Device Information;
STRUCT
    PV_AlarmSelCode   : BYTE;    //PV Alarm Selection Code;
    PV_TransFuncCode  : BYTE;    //PV Transfer Function Code;
    PV_UL_UnitCode    : BYTE;    //PV Upper and Lower Range Values Units Code;
    PV_UpperRangeValue : REAL;   //PV Upper Range Value;
    PV_LowerRangeValue : REAL;   //PV Lower Range Value;
    PV_DampingValue   : REAL;    //PV Damping Value;
    WriteProtectCode  : BYTE;    //Write Protect Code;
    PrivLblDistrCode  : BYTE;    //Private Label Distributor Code;
    PV_AnalogChFlags  : BYTE;    //PV Analog Channel Flags;
END_STRUCT
END_TYPE
```

Далее, в области объявлений программного блока (POU) обработки HART-команды следует объявить набор рабочих переменных.

```
//экземпляр FB пользовательского запроса;
UserRequest : PsIoDrvHartMaster.HartUserRequest;
//код команды;
bUR_Command : BYTE := 0;
```

```

//число попыток выполнения команды;
bUR_RepeatNum      : BYTE := 4;
//выходной параметр ФБ - флаг успешного выполнения команды;
xUR_Done           : BOOL := FALSE;
//выходной параметр ФБ - флаг ошибки выполнения команды;
xUR_Error          : BOOL := FALSE;
//размер структуры данных ответа;
bUR_CommandRxDataLen: BYTE;
//размер структуры данных запроса;
bUR_CommandTxDataLen: BYTE;

//флаг управления запуска/остановки команды;
xUR_Execute_Uni_15 : BOOL := FALSE;
//структура команды;
cmdUR_Command_15   : cmd15;
//формат блока данных ответа от устройства, содержащий размер полей данных в
байтах;
arrbUR_CommandRxFrmt_15: ARRAY[0..9] OF BYTE := [1, 1, 1, 4, 4, 4, 1, 1, 1,
0];
//количество полей блока данных ответа, включая завершающий '0';
bUR_CommandRxFrmtLen_15: BYTE := 10;
//формат блока данных запроса к устройству, содержащий размер полей данных в
байтах;
arrbUR_CommandTxFrmt_15: POINTER TO BYTE; //для этой команды не используется;
//количество полей блока данных запроса, включая завершающий '0';
bUR_CommandRTxFrmtLen_15: BYTE; //для этой команды не
используется;

```

В теле блока обработка команды сводится к следующим строкам:

```

// Выполнение пользовательского запроса
IF xUR_Execute_Uni_15 = TRUE THEN
    bUR_Command := 15;
    bUR_CommandRxDataLen := UINT_TO_BYTE(SIZEOF(cmdUR_Command_15.data));
    bUR_CommandTxDataLen := 0;
    UserRequest(
        xExecute := xUR_Execute_Uni_15,
        refHartOuterSlave := Hart_Outer_Slave,
        byCommand := bUR_Command,
        byRepeatNum := bUR_RepeatNum,
        sSpecificName := 'Test',
        pbyRData := ADR(cmdUR_Command_15.data),
        byRDataLen := bUR_CommandRxDataLen,
        pbyRDataFormat := ADR(arrbUR_CommandRxFrmt_15),
        byRDataFormatLen := bUR_CommandRxFrmtLen_15,
        bRespCode => cmdUR_Command_15.RespCode,
        bDevStatus => cmdUR_Command_15.DevStatus,
        eStatus => cmdUR_Command_15.Status,
        xDone => xUR_Done,
        xError => xUR_Error
    );
    IF xUR_Done OR xUR_Error THEN
        xUR_Execute_Uni_15 := FALSE;
        UserRequest(
            xExecute := xUR_Execute_Uni_15);
    END_IF
END_IF

```

В данном случае Hart\_Outer\_Slave – это имя экземпляра функционального блока, соответствующего устройству Hart Outer Slave в дереве устройств.

## Использование команды из контейнера команд HART-устройства

Для использования команды из контейнера команд HART-устройства необходимо создать в дереве устройств фиктивный контейнер и добавить/импортировать в него 15-ю команду, для которой автоматически создается тип структуры HartUserRequest\_Uni\_15 (Рисунок 30).

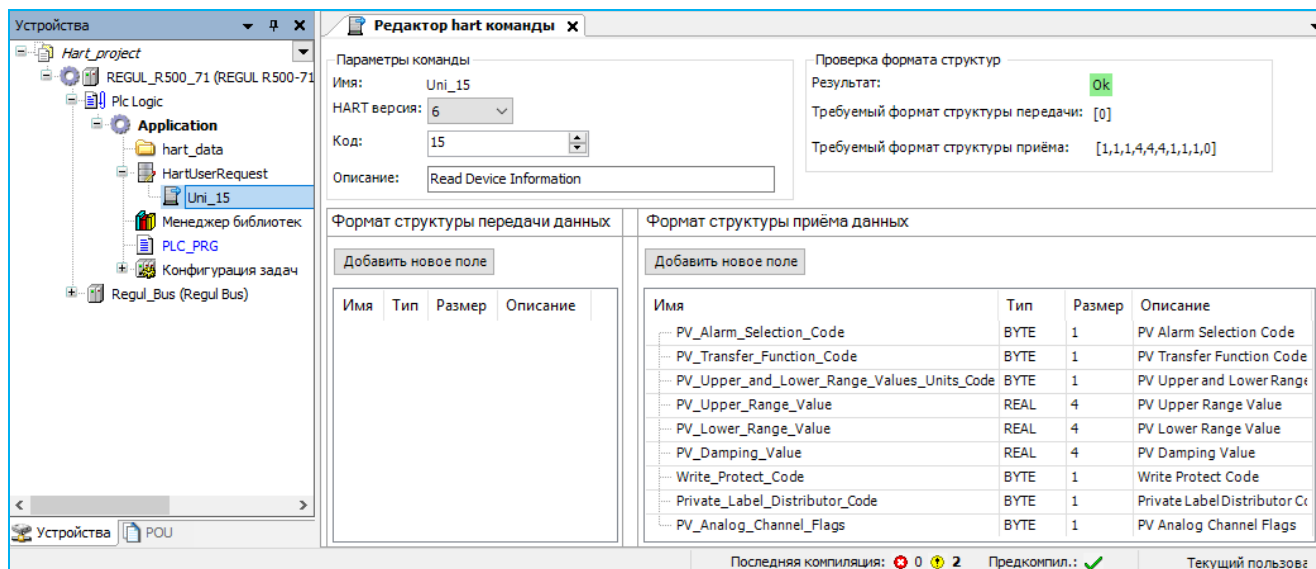


Рисунок 30 – Структура HartUserRequest\_Uni\_15

Далее, в области объявлений программного блока (POU) обработки HART-команды следует объявить набор рабочих переменных.

```
//экземпляр FB пользовательского запроса;
UserRequest          : PsIoDrvHartMaster.HartUserRequest;
//код команды;
bUR_Command          : BYTE := 0;
//число попыток выполнения команды;
bUR_RepeatNum        : BYTE := 4;
//выходной параметр ФБ – флаг успешного выполнения команды;
xUR_Done              : BOOL := FALSE;
//выходной параметр ФБ – флаг ошибки выполнения команды;
xUR_Error             : BOOL := FALSE;
//размер структуры данных ответа;
bUR_CommandRxDataLen: BYTE;
//размер структуры данных запроса;
bUR_CommandTxDataLen: BYTE;

//структура команды;
cmd_15                : HartUserRequest_Uni_15;
//формат блока данных ответа от устройства, содержащий размер полей данных в
байтах;
arrbUR_CommandRxFrm_15: ARRAY[0..9] OF BYTE := [1, 1, 1, 4, 4, 4, 1, 1, 1,
0];
//количество полей блока данных ответа, включая завершающий '0';
bUR_CommandRxFrmLen_15:  BYTE := 10;
//формат блока данных запроса к устройству, содержащий размер полей данных в
байтах;
arrbUR_CommandTxFrm_15: POINTER TO BYTE; //для этой команды не используется;
//количество полей блока данных запроса, включая завершающий '0';
bUR_CommandRTxFrmLen_15: BYTE;           //для этой команды не используется;
```

В теле блока обработка команды сводится к следующим строкам:

```
// Выполнение пользовательского запроса
IF cmd_15.CmdTrigger = TRUE THEN
    bUR_Command := 15;
    bUR_CommandRxDataLen := UINT_TO_BYTE(SIZEOF(cmd_15.rx_data));
    bUR_CommandTxDataLen := 0;
    UserRequest(
        xExecute := cmd_15.CmdTrigger,
        refHartOuterSlave := Hart_Outer_Slave,
        byCommand := bUR_Command,
        byRepeatNum := bUR_RepeatNum,
        sSpecificName := 'Test',
        pbyRData := ADR(cmd_15.rx_data),
        byRDataLen := bUR_CommandRxDataLen,
        pbyRDataFormat := ADR(arrbUR_CommandRxFrm_15),
        byRDataFormatLen := bUR_CommandRxFrmtLen_15,
        bRespCode => cmd_15.CmdRespCode,
        bDevStatus => cmd_15.CmdDevStatus,
        eStatus => cmd_15.CmdStatus,
        xDone => xUR_Done,
        xError => xUR_Error
    );
    IF xUR_Done OR xUR_Error THEN
        cmd_15.CmdTrigger := FALSE;
        UserRequest(
            xExecute := cmd_15.CmdTrigger);
    END_IF
END_IF
```

## Пример выполнения команды 34 – Write Primary Variable Damping Value

Для объявления структуры данных команды непосредственно в коде необходимо наследовать структуру 34-й команды от базовой структуры CmdBase, объявленной в библиотеке.

```
//COMMON PRACTICE 34 - Write Primary Variable Damping Value;
TYPE cmd34 EXTENDS PsIoDrvHartMaster.CmdBase:
STRUCT
    xdata : cmd34_data; //COMMON PRACTICE 34 - OUT - Write Primary Variable
Damping Value;
    rdata : cmd34_data; //COMMON PRACTICE 34 - IN - Write Primary Variable
Damping Value;
END_STRUCT
END_TYPE
```

Где:

```
{attribute 'pack_mode' := '1'}
TYPE cmd34_data : //COMMON PRACTICE 34 - Write Primary Variable Damping Value;
STRUCT
    PV_DampungValue : REAL;//Primary Variable Damping Value (units of seconds);
END_STRUCT
END_TYPE
```

Далее, в области объявлений программного блока (POU) обработки HART-команды следует объявить набор рабочих переменных.

```
//экземпляр FB пользовательского запроса;
UserRequest : PsIoDrvHartMaster.HartUserRequest;
//код команды;
```

```

bUR_Command                : BYTE := 0;
//число попыток выполнения команды;
bUR_RepeatNum              : BYTE := 4;
//выходной параметр ФБ - флаг успешного выполнения команды;
xUR_Done                   : BOOL := FALSE;
//выходной параметр ФБ - флаг ошибки выполнения команды;
xUR_Error                   : BOOL := FALSE;
//размер структуры данных ответа;
bUR_CommandRxDataLen      : BYTE;
//размер структуры данных запроса;
bUR_CommandTxDataLen      : BYTE;

//флаг управления запуска/остановки команды;
xUR_Execute_Cmn_34 : BOOL := FALSE;
//структура команды;
cmdUR_Command_34 : cmd34;
//формат блока данных ответа от устройства, содержащий размер полей данных в
байтах;
arrbUR_CommandRxFrmt_34 : ARRAY[0..1] OF BYTE := [4, 0];
//количество полей блока данных ответа, включая завершающий '0';
bUR_CommandRxFrmtLen_34 : BYTE := 2;
//формат блока данных запроса к устройству, содержащий размер полей данных в
байтах;
arrbUR_CommandTxFrmt_34 : ARRAY[0..1] OF BYTE := [4, 0];
//количество полей блока данных запроса, включая завершающий '0';
bUR_CommandTxFrmtLen_34 : BYTE := 2;

```

В теле блока обработка команды сводится к следующим строкам:

```

IF xUR_Execute_Cmn_34 = TRUE THEN
  bUR_Command := 34;
  bUR_CommandRxDataLen := UINT_TO_BYTE(SIZEOF(cmdUR_Command_34.rdata));
  bUR_CommandTxDataLen := UINT_TO_BYTE(SIZEOF(cmdUR_Command_34.xdata));
  cmdUR_Command_34.xdata.PV_DampungValue := 4.0;
  UserRequest(
    xExecute := xUR_Execute_Cmn_34,
    refHartOuterSlave := Hart_Outer_Slave,
    byCommand := bUR_Command,
    byRepeatNum := bUR_RepeatNum,
    sSpecificName := 'Test',
    pbyRData := ADR(cmdUR_Command_34.rdata),
    byRDataLen := bUR_CommandRxDataLen,
    pbyRDataFormat := ADR(arrbUR_CommandRxFrmt_34),
    byRDataFormatLen := bUR_CommandRxFrmtLen_34,
    byXDataLen := bUR_CommandTxDataLen,
    pbyXData := ADR(cmdUR_Command_34.xdata),
    pbyXDataFormat := ADR(arrbUR_CommandTxFrmt_34),
    byXDataFormatLen := bUR_CommandTxFrmtLen_34,
    bRespCode => cmdUR_Command_34.RespCode,
    bDevStatus => cmdUR_Command_34.DevStatus,
    eStatus => cmdUR_Command_34.Status,
    xDone => xUR_Done,
    xError => xUR_Error
  );
  IF xUR_Done OR xUR_Error THEN
    xUR_Execute_Cmn_34 := FALSE;
    UserRequest(
      xExecute := xUR_Execute_Cmn_34);
  END_IF
END_IF

```