

**Федеральное агентство по образованию  
Российской Федерации  
ГОУ ВПО «Российский химико-технологический университет  
им. Д.И. Менделеева»**

**Новомосковский институт (филиал)**

# **Проектирование АСР уровня с использованием языка Structured Text**

**Учебно–методическое пособие**

Утверждено Редакционно–издательским советом  
института в качестве методического пособия

**Новомосковск  
2010**

УДК 62-52  
ББК  
П

Рецензенты:

кандидат технических наук, доцент Предместьин В.Р.  
(НИ РХТУ им. Д. И. Менделеева)  
кандидат технических наук, инженер метролог Быков А.П.  
(ООО "НИАП-КАТАЛИЗАТОР")

Составители: Лопатин А. Г., Киреев П.А.

П 791 «**Проектирование АСР уровня с использованием языка Structured Text**» Учебно–методическое пособие /ГОУ ВПО «РХТУ им. Д. И. Менделеева», Новомосковский институт  
Сост.: Лопатин А. Г., Киреев П.А. Новомосковск, 2010.–113 с.

Представлены основы языка программирования SCADA-систем Structured Text. Рассмотрен пример создания АСР уровня на SCADA-системе Trace Mode с использованием языка программирования Техно ST.

Предназначено для студентов, обучающихся по специальности 210200 Автоматизация производственных процессов, а так же будет полезна всем лицам занимающиеся разработкой систем управления с использованием SCADA систем.

Ил. 143. Библиогр.: 4 назв.

УДК  
ББК

© ГОУ ВПО «Российский химико–технологический  
университет им. Д. И. Менделеева»,  
Новомосковский институт, 2010

## СОДЕРЖАНИЕ

Введение .....	4
Описание языков программирования стандарта МЭК 61131-3.....	5
Лабораторная работа .....	13
Разработка АСР на языке ST .....	13
Цель работы.....	13
Задание.....	13
Порядок работы .....	15
Содержание отчёта .....	88
Варианты.....	89
Вопросы для защиты .....	90
Список использованных источников.....	91
Приложение 1 .....	92
Пример оформления титульного листа лабораторной работы .....	92
Приложение 2.....	93
Описание Техно ST.....	93

## ВВЕДЕНИЕ

При создании АСУТП любой сложности всегда существовала тяжело решаемая проблема разработки ПО для рабочих мест операторов, то есть для верхнего уровня АСУТП и программирования промышленных контроллеров и интеллектуальных датчиков производилось иными программными средствами или специальными программами. Это было не очень удобно, но в условиях большого рыночного разнообразия процессоров и шин, неустановившихся стандартов, использование специфических программаторов казалось единственным выходом.

Но ситуация изменилась с момента массового распространения IBM PC совместимых компьютеров. Появилась возможность унифицировать ПО для операторских станций и промышленных контроллеров. В результате этой деятельности появились программные пакеты для создания человеко-машинного интерфейса (Human Machine Interface HMI) и программного обеспечения операторских станций АСУТП (Super Visor Control And Data Acquisition SCADA) – контроль управления и контроль сбора данных.

TRACE MODE - это программный пакет для разработки проектов автоматизации любой сложности. Это могут быть как небольшие технологические установки, так и крупные объекты, распределенные по большой территории, реализующие контроль и управление десятками тысяч параметров.

Проект TRACE MODE включает в себя программное обеспечение (ПО) для всех используемых АРМ и контроллеров – узлов проекта. Математические и графические компоненты их программного обеспечения одновременно загружаются в редакторы. Это делает проект прозрачным для разработчика и облегчает настройку взаимодействия узлов проекта и обмена данными.

## Описание языков программирования стандарта МЭК 61131-3

На современном этапе в качестве ядра любой системы промышленной автоматизации используется программируемый логический контроллер (ПЛК), к которому со стороны объекта автоматизации подключаются датчики и исполнительные органы. Через датчики в ПЛК поступает информация о текущем состоянии объекта, а через исполнительные органы ПЛК может изменять состояние управляемого объекта. Эта базовая схема может усложняться. Например, ПЛК могут подключаться к АРМ оператора для супервизорного управления или к БД для накопления информации и интеграции в АСУ предприятия. Поскольку все ПЛК строятся на базе цифровой техники, естественным образом предполагаются некоторые языковые средства их программирования. Причем в силу специфики задачи алгоритмические языки программирования, такие как Си, Паскаль, Си++, не годятся для этих целей.

В 1993 г. Международная электротехническая комиссия выпустила в свет стандарт МЭК 61131-3. Этот международный стандарт входит в группу МЭК 61131 стандартов, которые охватывают различные аспекты использования ПЛК. Декларируемые цели МЭК 61131-3 – стандартизация существующих языков ПЛК, а вернее, базовая платформа для такой работы в национальных комитетах стандартизации.

Специфика автоматизации предполагает наличие собственно системы управления, включающей датчики обратной связи и органы управления, и внешней (по отношению к системе управления) среды, на которую система управления воздействует через органы управления, – *объекта управления* – технической системы, реализующей некоторую производственную технологию. Воздействия – или, другими словами, *реакция* системы управления – определяются алгоритмом управления в зависимости от *событий* на объекте управления, информация о которых поступает через датчики обратной связи. Для цифровых систем это обстоятельство обуславливает *цикличность* управляющего алгоритма по схеме: считывание состояния входных сигналов через датчики – их обработка и формирование выходных сигналов – выдача выходных сигналов на исполнительные органы. *Событийность* предполагает алгоритмические изменения программы и набора обрабатываемых ею входных/выходных сигналов в зависимости от происходящих на объекте событий.

Алгоритм управления предполагает *синхронизацию* своего исполнения с физическими процессами во внешней среде, что обуславливает необходимость развитой службы времени и активную работу с временными объектами: задержками, паузами, таймаутами.

Другая характерная особенность алгоритмов управления – *логический параллелизм*, отражающий существование множества параллельно протекающих процессов в объекте управления. (Поскольку события, происходящие в различных компонентах системы, возникают независимо и в произвольной последовательности, то попытка задать реакцию системы

единым блоком означает комбинаторный перебор большого числа вариантов и неоправданный рост сложности описания). Логический параллелизм предполагает наличие в алгоритме управления независимых или слабо зависимых частей – логически обособленных потоков управления.

Поскольку программы пишутся человеком и исключительно для человека, то в силу особенностей человеческой психики языки должны быть просты в изучении. Кроме того, языки должны предоставлять *механизмы структуризации* алгоритма (в нашем случае – языковые средства организации совместного функционирования логически параллельных частей) и *механизмы абстрагирования* (в нашем случае – понятийный переход от датчиков и исполнительных органов к целевому технологическому процессу). Т.е. программа должна быть организована в виде обзримых, информационно-изолированных компонентов, возможно иерархически вложенных друг в друга, и на некотором уровне иерархии программирование должно вестись в естественных терминах технологического процесса. Перечисленные обстоятельства обуславливают разработку специализированных языков промышленной автоматизации.

В силу того, что общепринятого подхода к программированию ПЛК не существовало (и не существует до сих пор), членам комиссии не удалось договориться о едином языке. Поэтому было принято компромиссное решение – включить в стандарт языки, используемые в фирмах, представителям которых посчастливилось оказаться в членах группы. Среди языков-«счастливицев» оказались:

### **Язык LD**

Язык LD (Ladder Diagram - Принципиальная Схема) является графическим языком разработки, программа на котором представляет собой аналог релейной схемы. Пример программы на данном языке приведен на рис. 1.1. По идеи авторов стандарта, такая форма представления программы облегчит переход инженеров из области релейной автоматики на ПЛК.

К недостаткам данного языка можно отнести то, что по мере увеличения количества «реле» в схеме она становится сложнее для интерпретации, анализа и отладки. Еще один недостаток языка LD заключается в следующем: язык, построенный по аналогии с релейными схемами, может быть эффективно использован только для описания процессов, имеющих дискретный (двоичный) характер; для обработки «непрерывных» процессов (с множеством аналоговых переменных) такой подход теряет смысл.

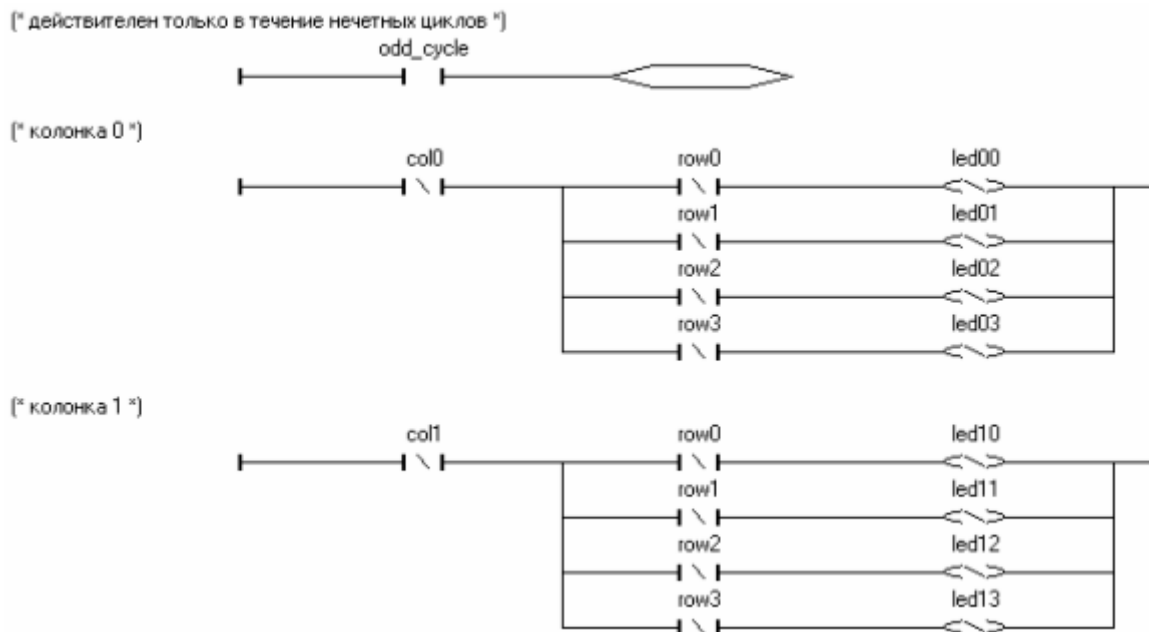


Рисунок 1.1 Язык релейных диаграмм LD.

### Язык FBD

Язык FBD (Functional Block Diagram - Функциональная Блок-схема) является языком графического программирования, так же, как и LD, использующий аналогию с электрической (электронной) схемой. Программа на языке FBD представляет собой совокупность функциональных блоков (functional blocks, FBs), входа и выхода которых соединены линиями связи (connections). Эти связи, соединяющие выхода одних блоков с входами других, являются по сути дела переменными программы и служат для пересылки данных между блоками. Каждый блок представляет собой математическую операцию (сложение, умножение, триггер, логическое “или” и т.д.) и может иметь, в общем случае, произвольное количество входов и выходов. Начальные значения переменных задаются с помощью специальных блоков – входов или констант, выходные цепи могут быть связаны либо с физическими выходами контроллера, либо с глобальными переменными программы. Пример фрагмента программы на языке FBD приведен на рис. 1.2.

Практика показывает, что FBD является наиболее распространенным языком стандарта ИЕС. Графическая форма представления алгоритма, простота в использовании, повторное использование функциональных диаграмм и библиотеки функциональных блоков делают язык FBD незаменимым при разработке программного обеспечения ПЛК. Вместе с тем, нельзя не заметить и некоторые недостатки FBD. Хотя FBD обеспечивает легкое представление функций обработки как «непрерывных» сигналов, в частности, функций регулирования, так и логических функций, в нем неудобным и неочевидным образом реализуются те участки программы, которые было бы удобно представить в виде конечного автомата.

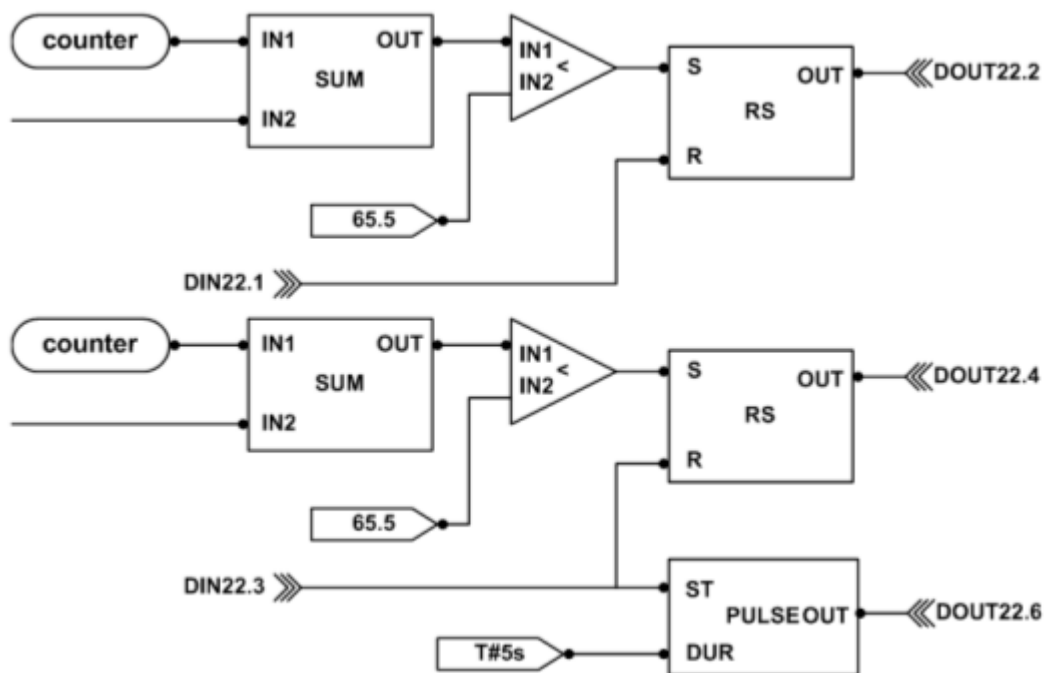


Рисунок 1.2 Функциональная схема FBD.

### Язык SFC

Язык SFC (Sequential Function Chart - Последовательная Функциональная Схема), использующийся совместно с другими языками (обычно с ST и IL), является графическим языком, в котором программа описывается в виде схематической последовательности шагов, объединенных переходами. Язык SFC построен по принципу, близкому к концепции конечного автомата, что делает его одним из самых мощных языков программирования стандарта IEC 61131-3. Пример программы на языке SFC приведен на рис. 1.3.

Наиболее простым и естественным образом на языке SFC описываются технологические процессы, состоящие из последовательно выполняемых шагов, с возможностью описания нескольких параллельно выполняющихся процессов, для чего в языке имеются специальные символы разветвления и слияния потоков (дивергенции и конвергенции, в терминах стандарта IEC 61131-3).

Шаги последовательности располагаются вертикально сверху вниз. На каждом шаге выполняется определенный перечень действий (операций). При этом для описания самой операции используются другие языки программирования, такие как IL или ST.

Действия (операции) в шагах имеют специальные классификаторы, определяющие способ их выполнения внутри шага: циклическое выполнение, однократное выполнение, однократное выполнение при входе в шаг и т.д. В сумме таких классификаторов насчитывается девять, причем среди них есть, например, классификаторы так называемых сохраняемых и отложенных действий, заставляющие действие выполняться даже после выхода программы из шага.

После того, как шаг выполнен, управление передается следующему за ним шагу. Переход между шагами может быть условным и безусловным. Условный переход требует выполнение определенного логического условия



для передачи управления на следующий шаг; пока это условие не выполнено программа будет оставаться внутри текущего шага, даже если все операции внутри шага уже выполнены. Безусловный переход происходит всегда после полного выполнения всех операций на данном шаге. С помощью переходов можно осуществлять разделение и слияние ветвей последовательности, организовать параллельную обработку нескольких ветвей или заставить одну выполненную ветвь ждать завершения другой.

Как и любому другому языку, SFC свойственны некоторые недостатки. Хотя SFC может быть использован для моделирования конечных автоматов, его программная модель не совсем удобна для этого. Это связано с тем, что текущее состояние программы определяется не переменной состояния, а набором флагов активности каждого шага, в связи с чем, при недостаточном контроле со стороны программиста могут оказаться одновременно активными несколько шагов, не находящихся в параллельных потоках.

Еще одно неудобство языка связано с тем, что шаги графически располагаются сверху вниз, и переход, идущий в обратном направлении, изображается в неявной форме, в виде стрелки с номером состояния, в которое осуществляется переход.

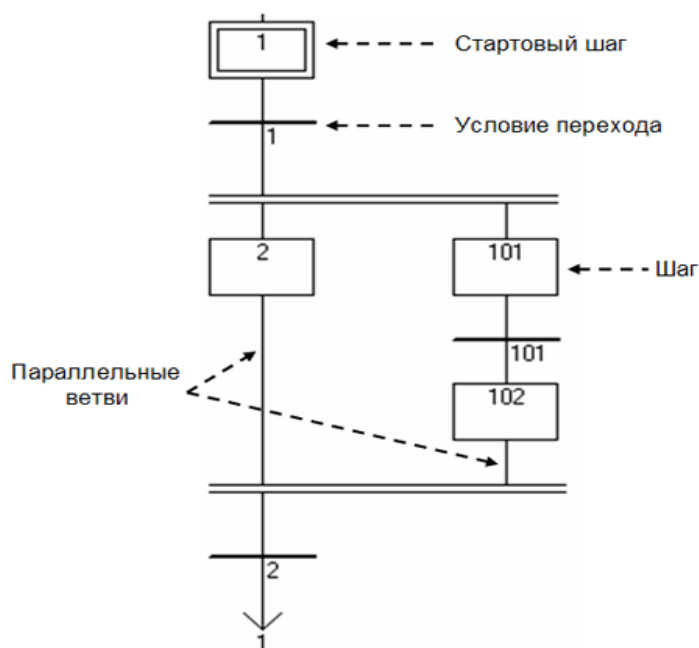


Рисунок 1.3. Язык последовательных функциональных схем SFC.

### Язык ST

Язык ST (Structured Text - Структурированный Текст) представляет собой язык высокого уровня, имеющий черты языков Pascal и Basic. Данный язык имеет те же недостатки, что и IL, однако они выражены в меньшей степени. Пример программы на языке ST приведен на рис. 1.4.

С помощью ST можно легко реализовывать арифметические и логические операции (в том числе, побитовые), безусловные и условные переходы, циклические вычисления; возможно использование как библиотечных, так и пользовательских функций. Язык также интерпретирует более 16 типов данных.

Язык ST может быть освоен технологом за короткий срок, однако текстовая форма представления программ служит сдерживающим фактором при разработке сложных систем, так как не дает наглядного представления ни о структуре программы, ни о происходящих в ней процессах.

```
blinker (TRUE, t#1s);  
trigger (blinker.q);  
  
if trigger.q then  
  counter := counter + 1;  
  if counter >= 4 then  
    counter := 0;  
  end_if;  
end_if;  
  
st0 := (counter = 0);  
st1 := (counter = 1);  
st2 := (counter = 2);  
st3 := (counter = 3);
```

Рисунок 1.4. Язык структурированного текста ST.

## Язык IL

Язык IL (Instruction List - Список Команд) представляет собой ассемблероподобный язык, достаточно несложный по замыслу авторов стандарта, для его практического применения в задачах промышленной автоматизации пользователем, не имеющим, с одной стороны, профессиональной подготовки в области программирования, с другой стороны, являющимся специалистом в той или иной области производства. Однако, как показывает практика, такой подход себя не оправдывает.

Ввиду своей ненаглядности, IL практически не используется для программирования комплексных алгоритмов автоматизированного управления, но часто применяется для кодирования отдельных функциональных блоков, из которых впоследствии складываются схемы FBD или SFC. При этом IL позволяет достичь высокой оптимальности кода: программные блоки, написанные на IL, имеют высокую скорость исполнения и наименее требовательны к ресурсам контроллера.

Язык IL имеет все недостатки, которые присущи другим низкоуровневым языкам программирования: сложность и высокую трудоемкость программирования, трудность модификации написанных на нем программ, малую степень «видимого» соответствия исходного текста программы и решаемой задачи.

Пример программы на языке IL приведен на рис. 1.5.

```

ld    true
st    blinker.run
ld    t#1s
st    blinker.cycle
cal   blinker

ld    blinker.q
st    trigger.clk
cal   trigger

ld    trigger.q
jmpnc LBmodulo
ld    counter
add   1
st    counter

LBmodulo:
ld    counter
lt    4
jmpc  LBout
ld    0
st    counter

LBout:
ld    counter
eq    0
st    il0
ld    counter
eq    1
st    il1
ld    counter
eq    2
st    il2
ld    counter
eq    3
st    il3

ret

```

Рисунок 1.5. Язык инструкций IL.

Многие производители инструментальных средств, опирающиеся на стандарт IEC, не ограничиваются поддержкой рассмотренных выше пяти языков стандарта. Можно выделить, как минимум, еще один язык визуального программирования, который довольно популярен среди разработчиков.

### Язык CFC

Язык CFC (Continuous Flow Chart - Непрерывная Блок-схема) – еще один высокоуровневый язык визуального программирования. По сути, CFC – это дальнейшее развития языка FBD. Этот язык был специально создан для проектирования систем управления непрерывными технологическими процессами.

Проектирование сводится к выбору из библиотек готовых функциональных блоков, их позиционированию на экране, установке соединений между их входами и выходами, а также настройке параметров выбранных блоков. В отличие от FBD, функциональные блоки языка CFC выполняют не только простые математические операции, а ориентированы на управление целыми технологическими единицами. Так в типовой библиотеке CFC блоков находятся комплексные функциональные блоки, реализующие управление клапанами, моторами, насосами; блоки, генерирующие аварийные сигнализации; блоки PID-регулирования и т.д. Вместе с тем доступны и стандартные блоки FBD. Унаследовав от FBD саму концепцию программирования, язык CFC в наибольшей степени ориентирован на сам

СФС прост в освоении, и при этом позволяет разрабатывать сложнейшие алгоритмы автоматизированного управления без каких-либо специфических знаний других языков программирования.

## ЛАБОРАТОРНАЯ РАБОТА

### РАЗРАБОТКА АСР НА ЯЗЫКЕ ST

**Цель работы:** изучения языка программирования SCADA-программ – Structured Text (структурированный текст) на примере разработки АСР уровня.

**Задание:**

1) Используя язык программирования Trace Mode - Техно ST реализовать модель ёмкости:

$$L(t) = \int_0^t \frac{F^{in}(t) - F^{out}(t)}{S} \cdot dt \quad (1.1)$$

где

$L(t)$  – уровень в ёмкости, см

$F^{in}(t)$  – расход жидкости, поступающей в ёмкость, см<sup>3</sup>/с

$F^{out}(t)$  – расход жидкости, уходящей из ёмкости, см<sup>3</sup>/с

$S$ , площадь ёмкости, см<sup>2</sup>

для удобства реализации нужно воспользоваться дискретной формулой:

$$L_i = \sum_{j=1}^i \frac{F_j^{in} - F_j^{out}}{S} \cdot \Delta t \quad (1.2)$$

из которой выводится рекуррентная:

$$L_i = L_{i-1} + \frac{F_i^{in} - F_i^{out}}{S} \cdot \Delta t \quad (1.3)$$

где

$L_i$  и  $L_{i-1}$  – значение уровня в ёмкости, соответственно в  $i$ -ый и  $i-1$  моменты времени, см

$F_i^{in}$  – расход жидкости, поступающей в ёмкость в  $i$ -ый моменты времени, см<sup>3</sup>/с

$F_i^{out}$  – расход жидкости, уходящей из ёмкости, см<sup>3</sup>/с

$S$ , площадь ёмкости, см<sup>2</sup>

$\Delta t$  – интервал времени между  $i$ -ым и  $i-1$  моментами времени, с

для примера будем полагать:

площадь  $S = 1 \text{ см}^2$ ,

интервал времени  $\Delta t = 1 \text{ с}$ ,

максимальный расход поступающей жидкости  $F_{\max}^{in} = 2 \text{ см}^3 / \text{с}$ ,

максимальный расход уходящей жидкости  $F_{\max}^{out} = 2 \text{ см}^3 / \text{с}$

максимальный уровень ёмкости  $L_{\max} = 200 \text{ см}$

минимальный уровень ёмкости  $L_{\min} = 0 \text{ см}$

2) используя язык программирования Trace Mode - Техно ST создать двухпозиционный регулятор с зоной нечувствительности рис. 1.7:

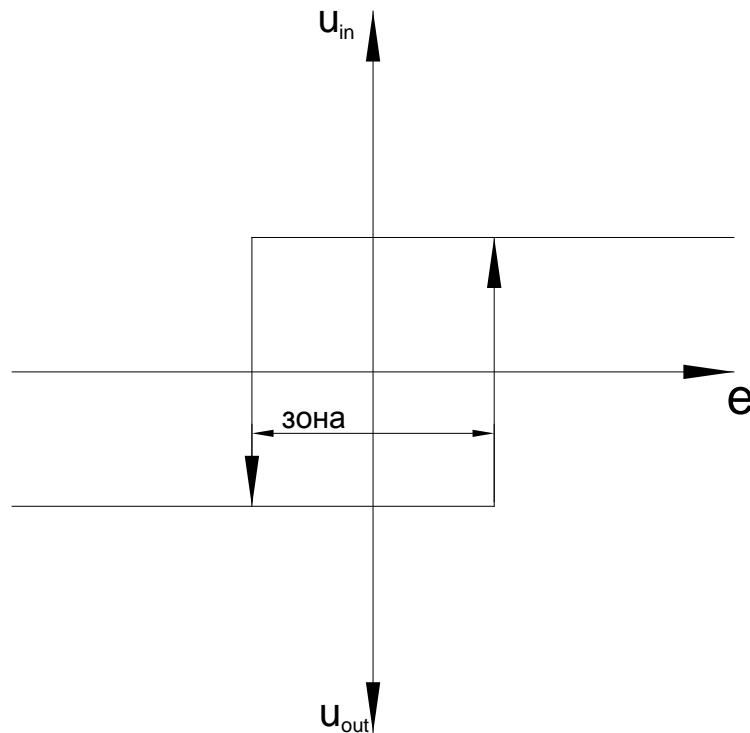


Рисунок 1.7 График работы двухпозиционного регулятора с зоной нечувствительности

где

$e$  - рассогласование между заданием и уровнем

$u_{in}$  - управляющее воздействие клапана регулирующего расход жидкости, поступающей в ёмкость

$u_{out}$  - управляющее воздействие клапана регулирующего расход жидкости, уходящей из ёмкости

*зона* – величина зоны нечувствительности позиционного регулятора

3) используя графические элементы Trace Mode, создать интерфейс автоматизированного рабочего места, в котором реализовано:

- схематическое отображение технологического процесса,
- отображение работы систем автоматики (исполнительных устройств)
- отображения информации об уровне,
- отображения информации изменение уровня во времени,
- установка задания,
- установка зоны нечувствительности регулятора

## ПОРЯДОК РАБОТЫ

Запустите SCADA-систему Trace Mode 6:

Левой клавишей (ЛК) мыши нажмите последовательно **Пуск->Все программы->Trace Mode 6 (base)-> Trace Mode 6 (base)** (рис. 2.1).

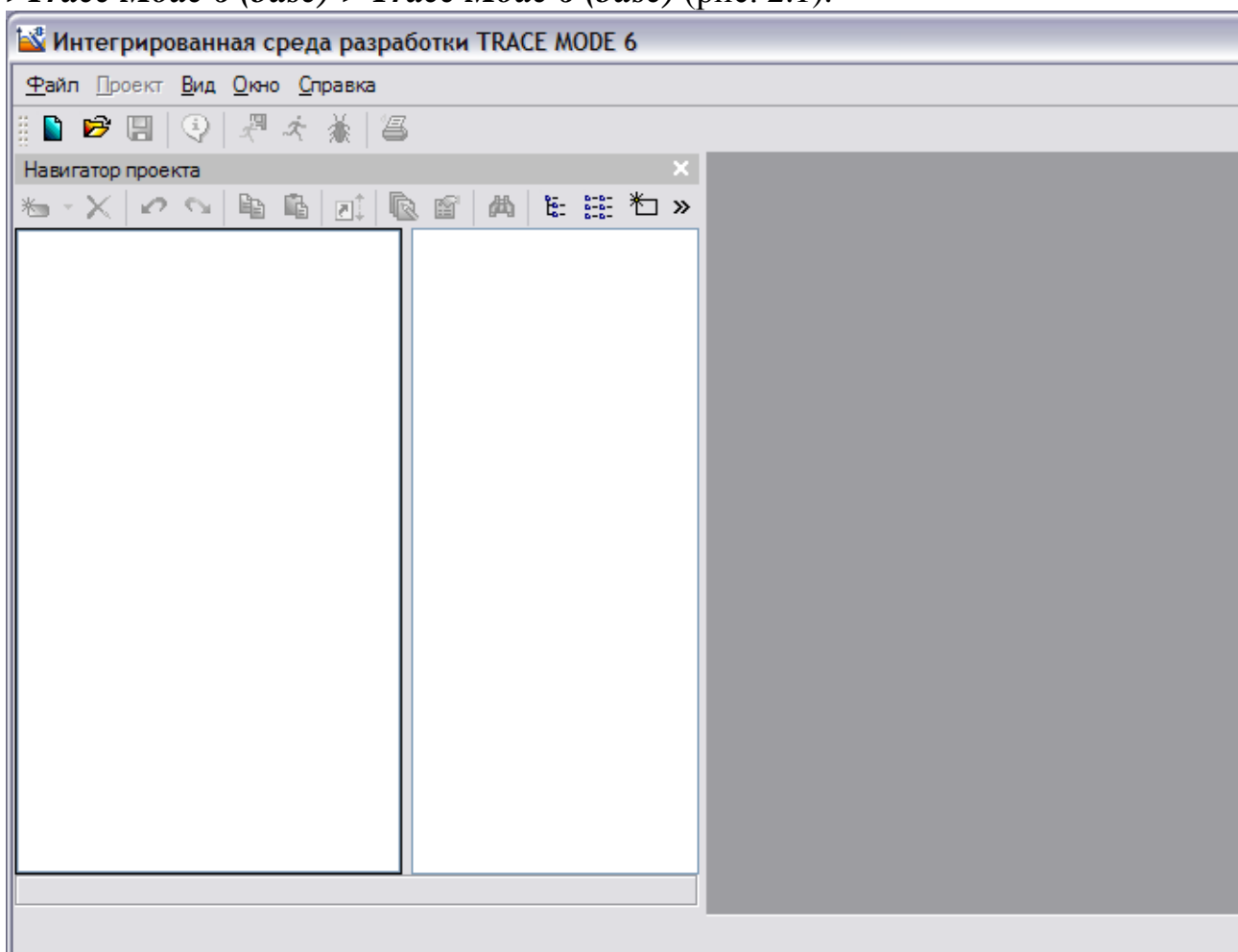


Рисунок 2.1. Интегрированная среда разработки

В открывшемся окне программы создайте новый проект:  
Меню **Файл->Новый** (рис. 2.2).

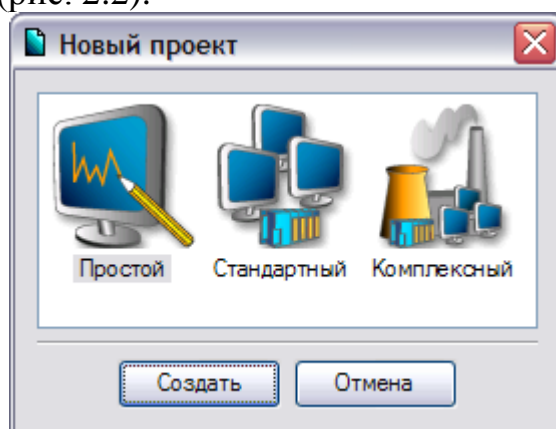


Рисунок 2.2. Новый проект

В окне **Новый проект** выберите стиль разработки **Простой** и нажмите ЛК мыши кнопку **Создать** (рис. 2.3).

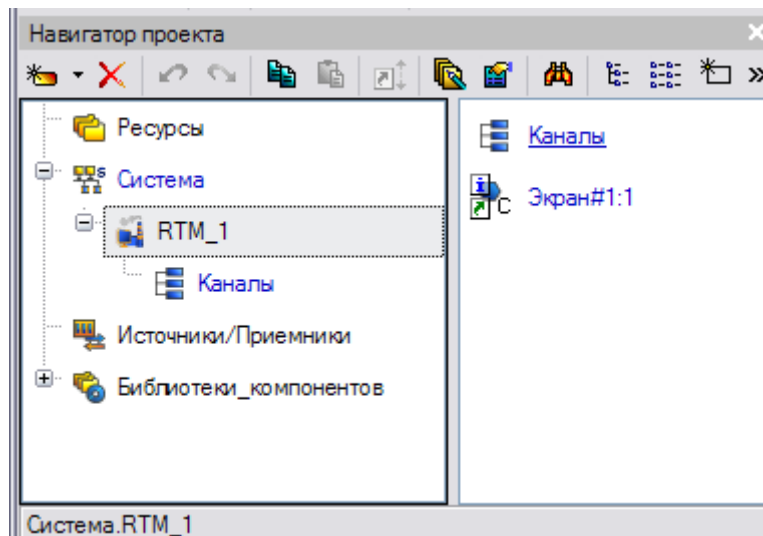


Рисунок 2.3. Навигатор проекта

В левом окне *Навигатора проекта* находится дерево проекта с созданным узлом АРМ **RTM\_1**. В правом окне Навигатора проекта отображается содержимое узла – пустая группа **Каналы** и один канал класса **Вызов - Экран#1**, предназначенный для отображения на узле АРМ графического экрана.

Первоначально надо создать программу эмулирующую работу ёмкости.

Для создания программы надо открыть узел **RTM\_1** и создать в нём компонент **Программа**: щелчком ПК вызвать контекстное меню и выбрать: *Создать компонент->Программа* (рис. 2.4).

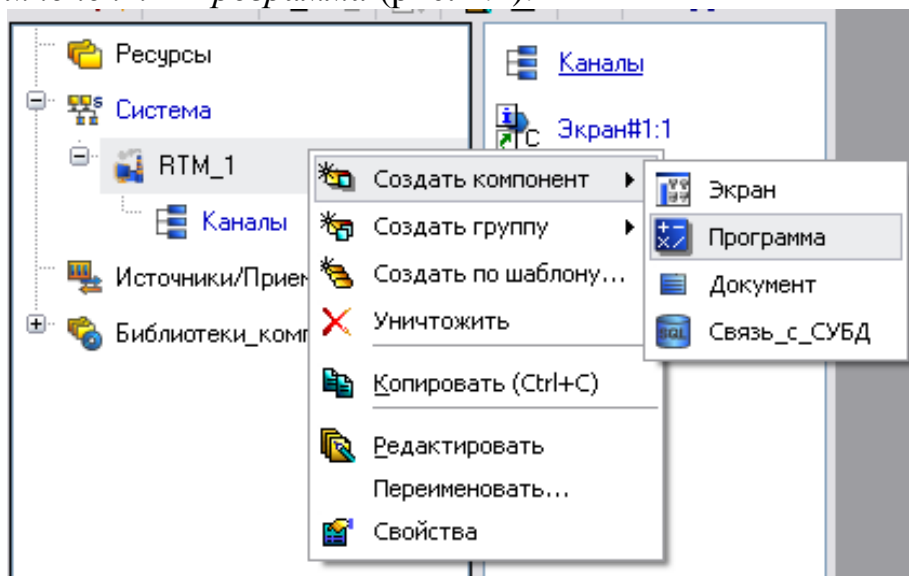


Рисунок 2.4. Контекстное меню узла – создание программы  
Двойным щелчком ЛК по компоненту **Программа#1** перейти в режим редактирования программы (рис. 2.5).



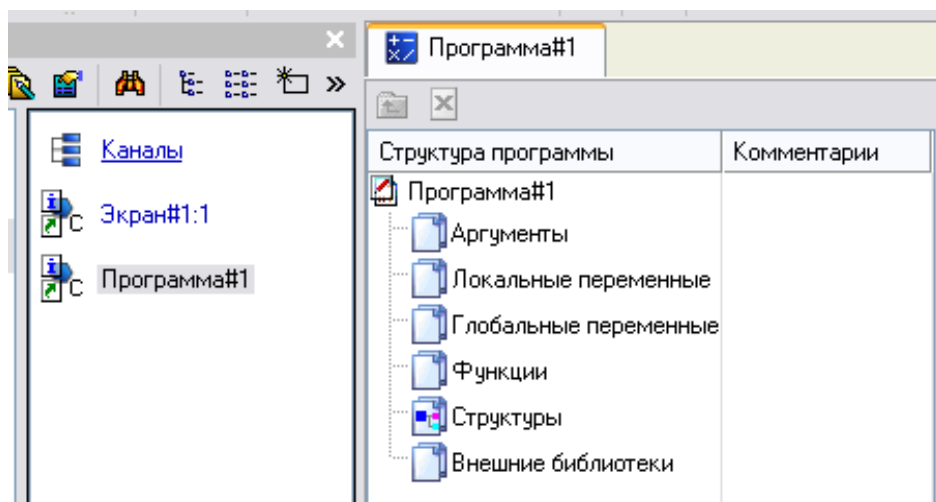


Рисунок 2.5. Структура программы

Выделив ЛК в дереве шаблона **Программа#1** строку **Аргументы**, вызвать табличный редактор документов (рис. 2.6)

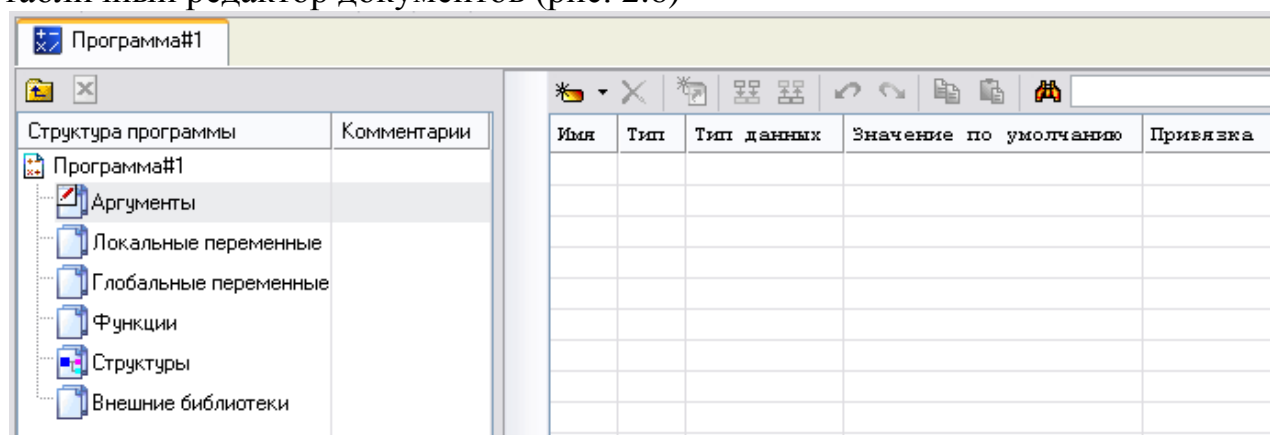


Рисунок 2.6. Редактор аргументов

С помощью иконки  создать в редакторе три аргумента:

- 1) *Залив* – тип **IN**,
- 2) *Слив* – тип **IN**,
- 3) *Уровень* – тип **Out**.

Для изменения имени аргумента нужно двойным щелчком ЛК зайти в ячейку **Имя** аргумента и внести необходимые изменения завершив ввод нажатием клавиши **Enter**.

Для смены типа аргумента необходимо двойным щелчком ЛК по ячейке **Тип** аргумента вызвать выпадающий список, в котором выбрать соответствующее значение (рис. 2.7)

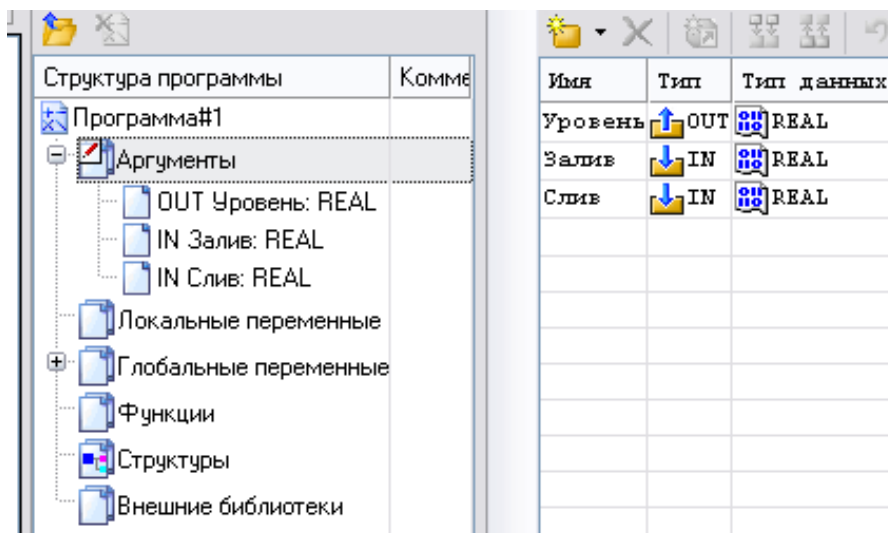



Рисунок 2.7. Создание аргументов

Выделив ЛК в дереве шаблона **Программа#1** строку **Глобальные переменные**, вызвать табличный редактор документов и с помощью иконки  создать в редакторе глобальную переменную *Level*, в которой будет накапливаться значение уровня (рис. 2.8).

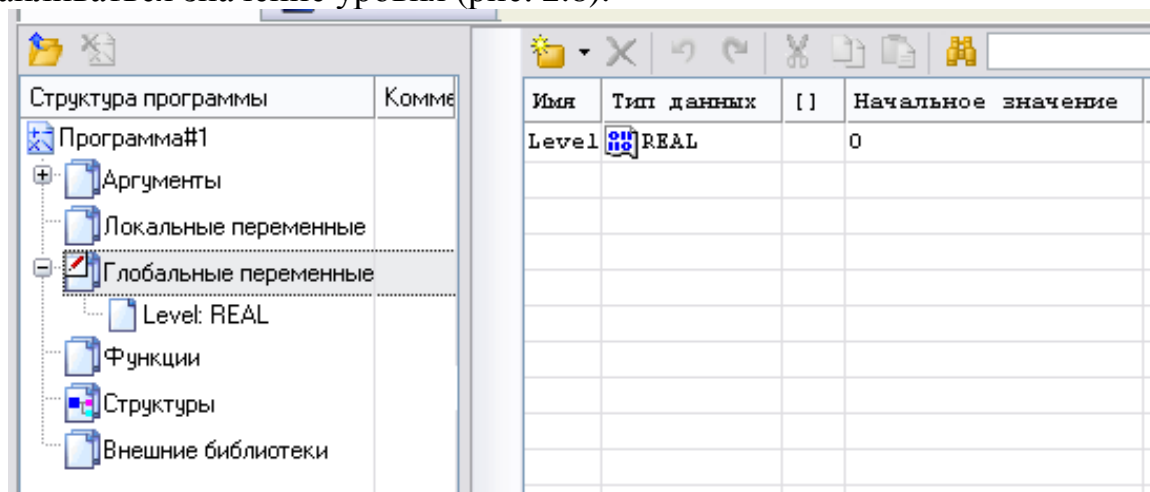


Рисунок 2.8. Создание глобальных переменных.

Выделив ЛК в дереве шаблонов (вкладка **Программа#1**) строку **Программа#1**, в открывшемся диалоге **Выбор языка** выбрать язык **ST** (рис. 2.9).

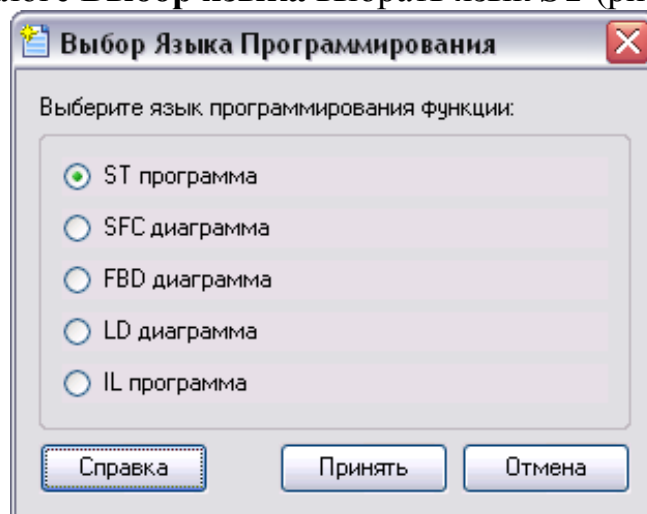
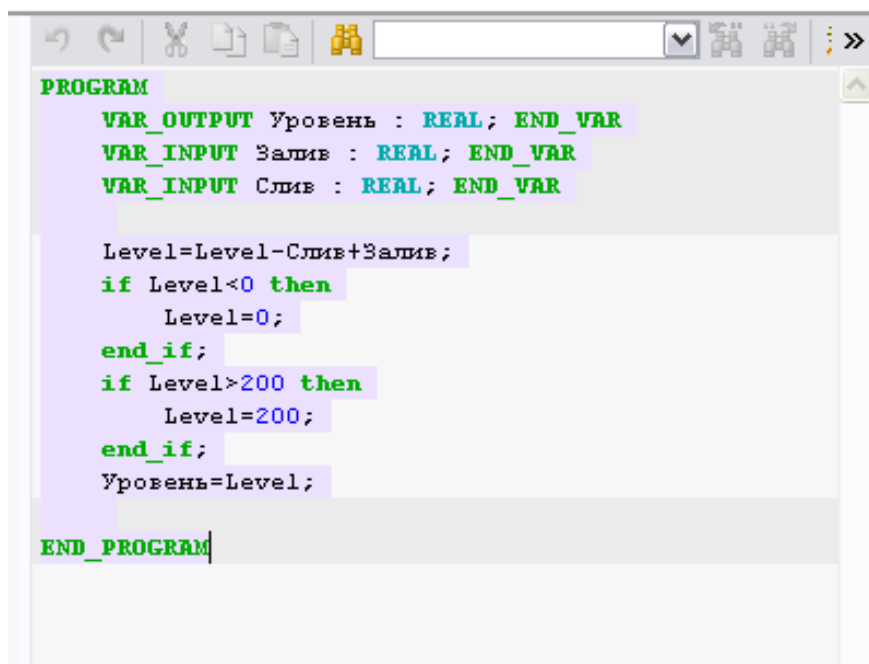


Рисунок 2.9. Выбор языка программирования

Нажать кнопку **Принять**.



На открывшейся справа панели, написать программу описывающую модель ёмкости (рис. 2.10)



```
PROGRAM
  VAR_OUTPUT Уровень : REAL; END_VAR
  VAR_INPUT Залив : REAL; END_VAR
  VAR_INPUT Слив : REAL; END_VAR

  Level=Level-Слив+Залив;
  if Level<0 then
    Level=0;
  end_if;
  if Level>200 then
    Level=200;
  end_if;
  Уровень=Level;
END_PROGRAM
```

Рисунок 2.10. Техно-ST программа модели ёмкости

С помощью иконки  на инструментальной панели редактора или нажатием "горячей клавиши" **F7** скомпилировать программу и убедиться в успешной компиляции в окне **Выход (Output)**, вызываемом из инструментальной панели с помощью иконки  (рис. 2.11)

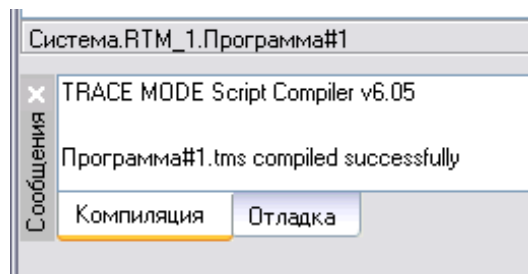


Рисунок 2.11. Результат компиляции программы

Следующим шагом необходимо создать двухпозиционный регулятор уровня с зоной нечувствительности.

Для создания программы надо открыть узел **RTM\_1** и создать в нём компонент **Программа**: щелчком ПК вызвать контекстное меню и выбрать: *Создать компонент->Программа* (рис. 2.12).

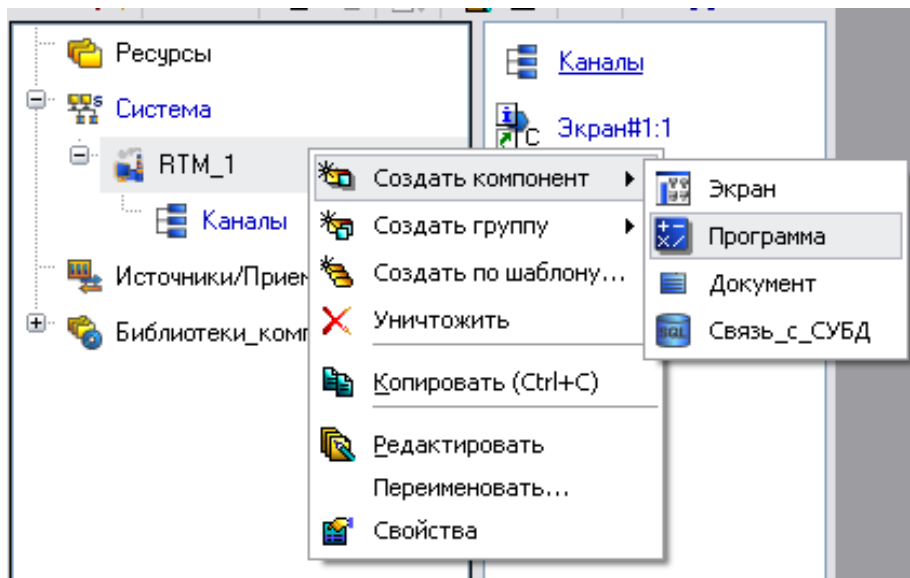


Рисунок 2.12. Контекстное меню – создание программы

Двойным щелчком ЛК по компоненту **Программа#2** перейти в режим редактирования программы. Затем, выделив ЛК в дереве шаблона **Программа#1** строку **Аргументы**, вызвать табличный редактор документов. С помощью иконки \* ▾ создать в редакторе пять аргументов (рис. 2.13):

- 1) *Задание* – тип **IN**,
- 2) *Зона* – тип **IN**,
- 3) *Залив* – тип **Out**,
- 4) *Слив* – тип **Out**,
- 5) *Уровень* – тип **IN**.

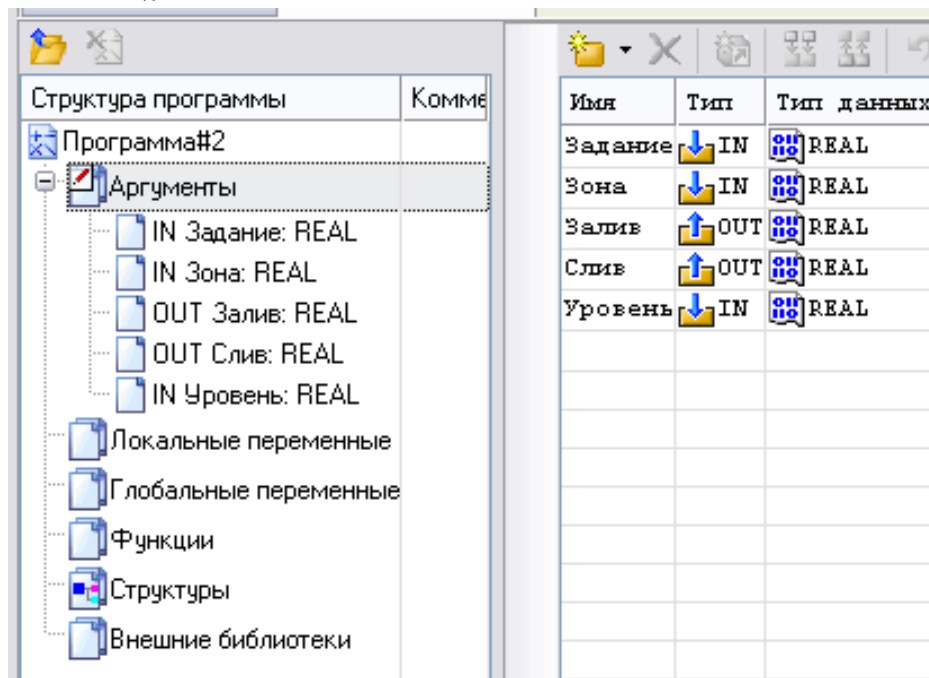


Рисунок 2.13. Создание аргументов

Выделив ЛК в дереве шаблонов (вкладка **Программа#2**) строку **Программа#2**, в открывшемся диалоге **Выбор языка** выбрать язык **ST** (рис. 2.14)

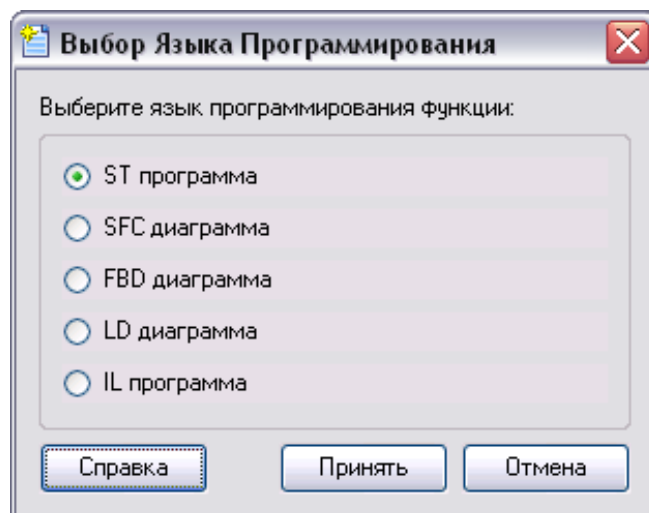


Рисунок 2.14. . Выбор языка программирования

Нажать кнопку **Принять**.

На открывшейся справа панели написать программу, описывающую двухпозиционного регулятора с зоной нечувствительности (рис. 2.15).

```



PROGRAM
VAR_INPUT Задание : REAL; END_VAR
VAR_INPUT Зона : REAL; END_VAR
VAR_OUTPUT Залив : REAL; END_VAR
VAR_OUTPUT Слив : REAL; END_VAR
VAR_INPUT Уровень : REAL; END_VAR

if Задание-Уровень > Зона then
    Залив=2;
    Слив=0;
end_if;
if Уровень-Задание > Зона then
    Залив=0;
    Слив=2;
end_if;
if Задание-Уровень < Зона && Уровень-Задание < Зона then
    Залив=0;
    Слив=0;
end_if;

END_PROGRAM

```

Рисунок 2.15 Техно-ST программа регулятора

С помощью иконки  на инструментальной панели редактора или нажатием "горячей клавиши" **F7** скомпилировать программу и убедиться в успешной компиляции в окне **Выход (Output)**, вызываемом из инструментальной панели с помощью иконки  (рис. 2.16).

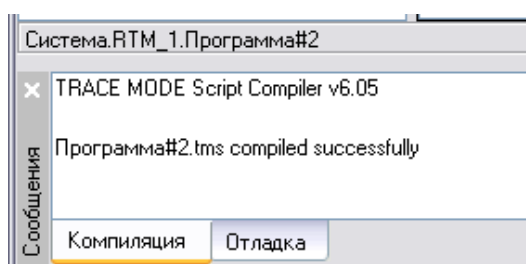


Рисунок 2.16. Результат компиляции программы

Далее надо создать графический интерфейс системы управления.

Двойным щелчком ЛК на компоненте **Экран#1:1** открыть окно графического редактора.



На панели инструментов графического редактора двойным щелчком ЛК по кнопке  вызвать панель **Объемные фигуры** (рис. 2.17).



Рисунок 2.17. Панель инструментов графического редактора – Объемные фигуры

На данной панели выбрать щелчком ЛК графический элемент (ГЭ) *Ёмкость* , затем, в поле графического редактора, ЛК указать левый верхний угол ГЭ, движением мыши растянуть до необходимого размера и зафиксировать правый нижний угол ЛК (рис. 2.18).

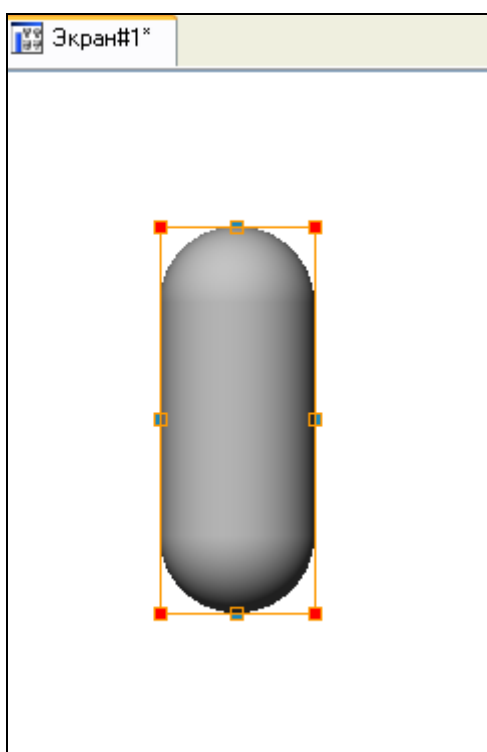


Рисунок 2.18. Графический элемент *Ёмкость*



На панели инструментов графического редактора двойным щелчком ЛК по кнопке  вызвать панель **Плоские фигуры** (рис. 2.19).



Рисунок 2.19. Панель инструментов графического редактора – Плоские фигуры

На панели выбрать ГЭ *Овал* -  и разместить его поверх ГЭ *Ёмкость* (рис. 2.20).

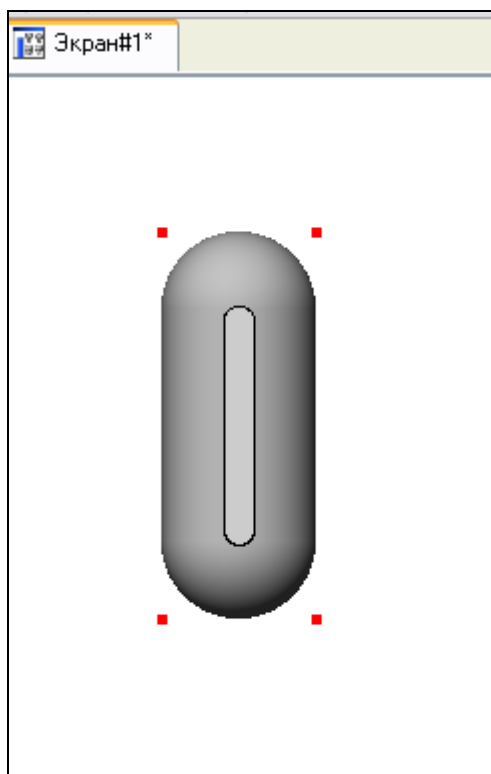



Рисунок 2.20. Ёмкость с индикатором уровня

Затем перейти в режим редактирования атрибутов ГЭ *Овал*, выделив ЛК иконку  на панели инструментов и двойным щелчком ЛК по ГЭ открыть его свойства (рис. 2.21).

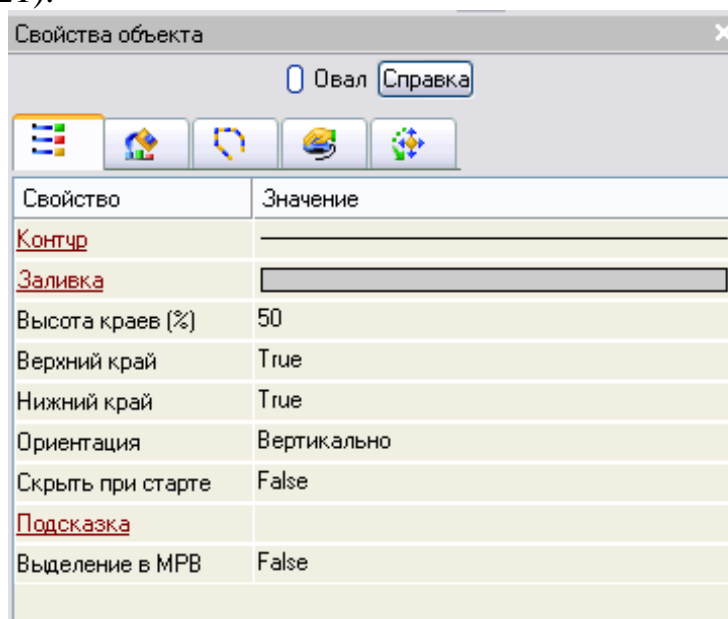


Рисунок 2.21. Свойства графического элемента *Овал*

Переключится ЛК на вкладку **Динамическая заливка** (рис. 2.22)

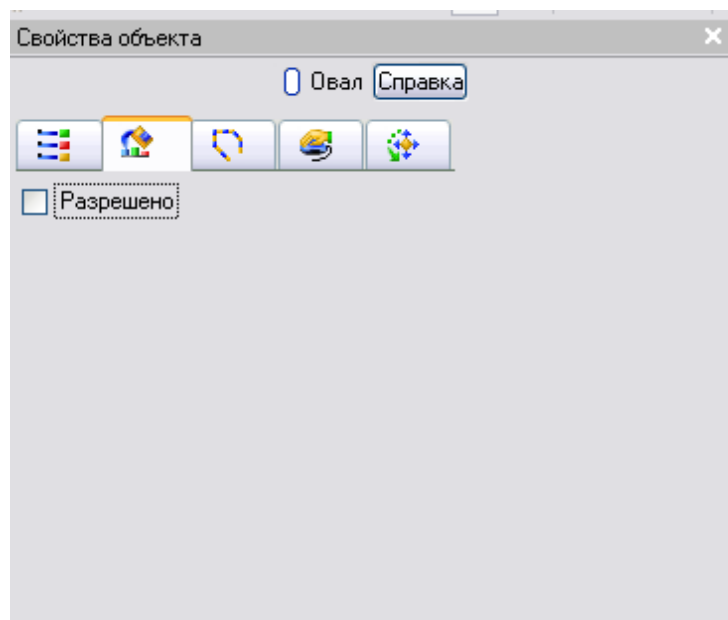


Рисунок 2.22. Свойство Свойства графического элемента *Овал* – Динамическая заливка

Поставить ЛК галочку **Разрешено** (рис. 2.23)

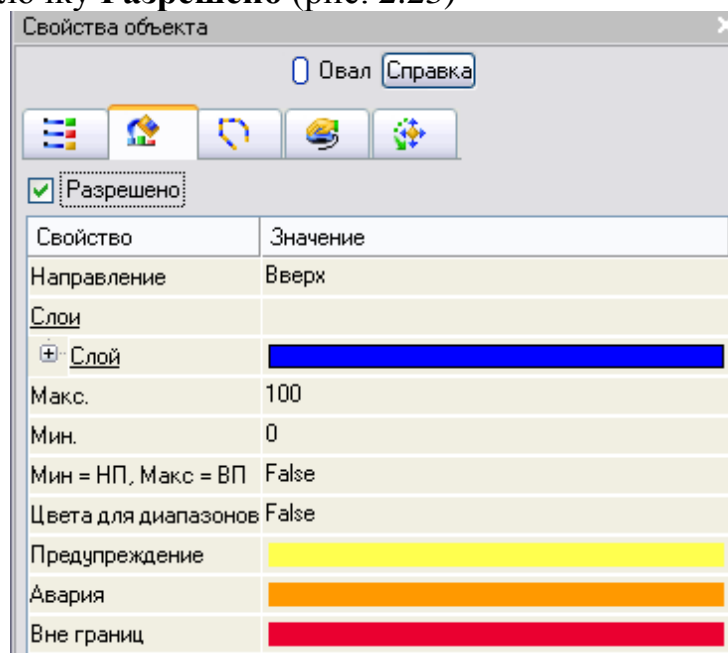


Рисунок 2.23. Свойство графического элемента *Овал* – Динамическая заливка - Слой

В появившемся свойстве *Слой* развернуть щелчком ЛК по знаку «+» меню **Слой** (рис. 2.24).





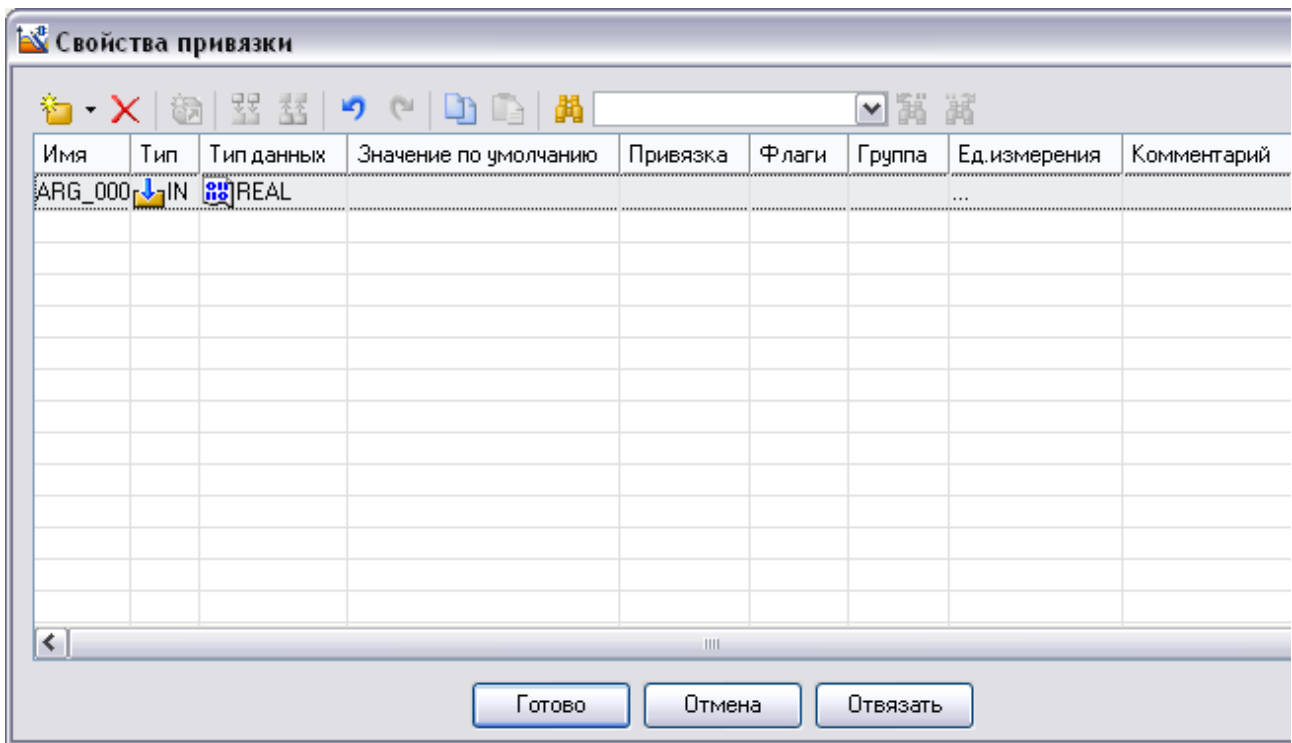


Рисунок 2.26. Свойства привязки – создание нового аргумента  
 Задать его тип **IN**, а затем связать его с аргументом **Программы#1 - Уровень**.  
 Для смены типа аргумента необходимо двойным щелчком ЛК по ячейке **Тип**  
 аргумента вызвать выпадающий список, в котором выбрать соответствующее  
 значение (рис. 2.27).

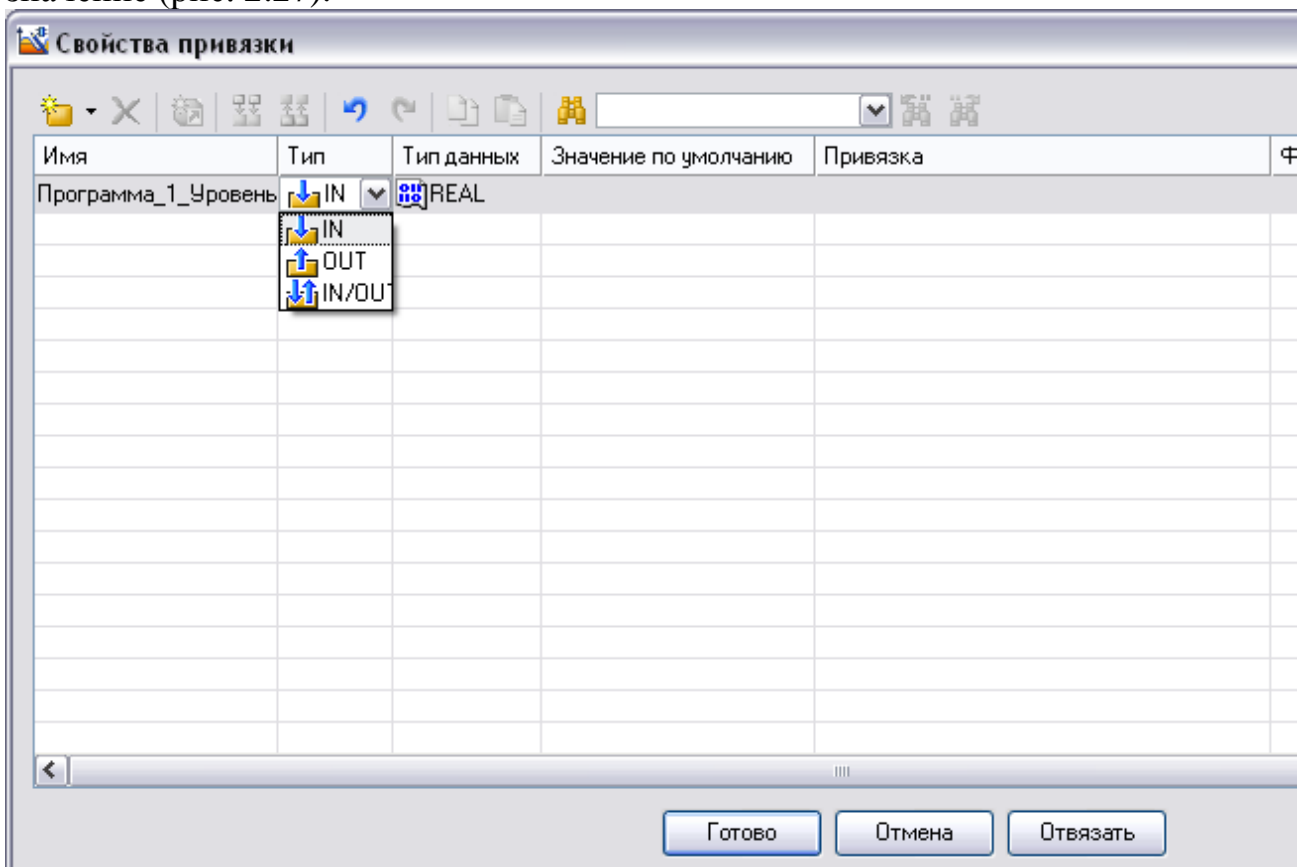


Рисунок 2.27. Свойства привязки – изменение типа аргумента

Для привязки аргумента двойным щелчком ЛК по ячейке **Привязка** аргумента вызвать окно конфигурации, в которой необходимо выбрать соответствующий аргумент - **Программы#1 - Уровень** (рис. 2.28).

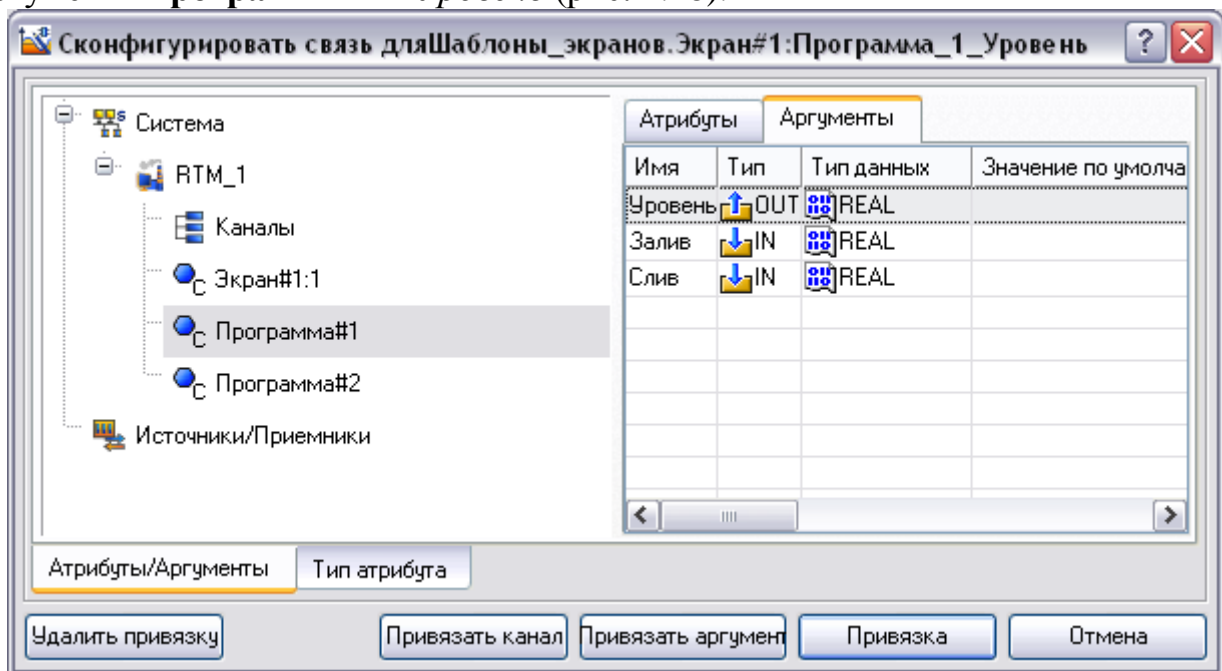


Рисунок 2.28. Привязка динамической заливки к аргументу *Уровень*  
Нажать кнопку **Привязка**.

После привязки аргумента нажать кнопку **Готово** (рис. 2.29).

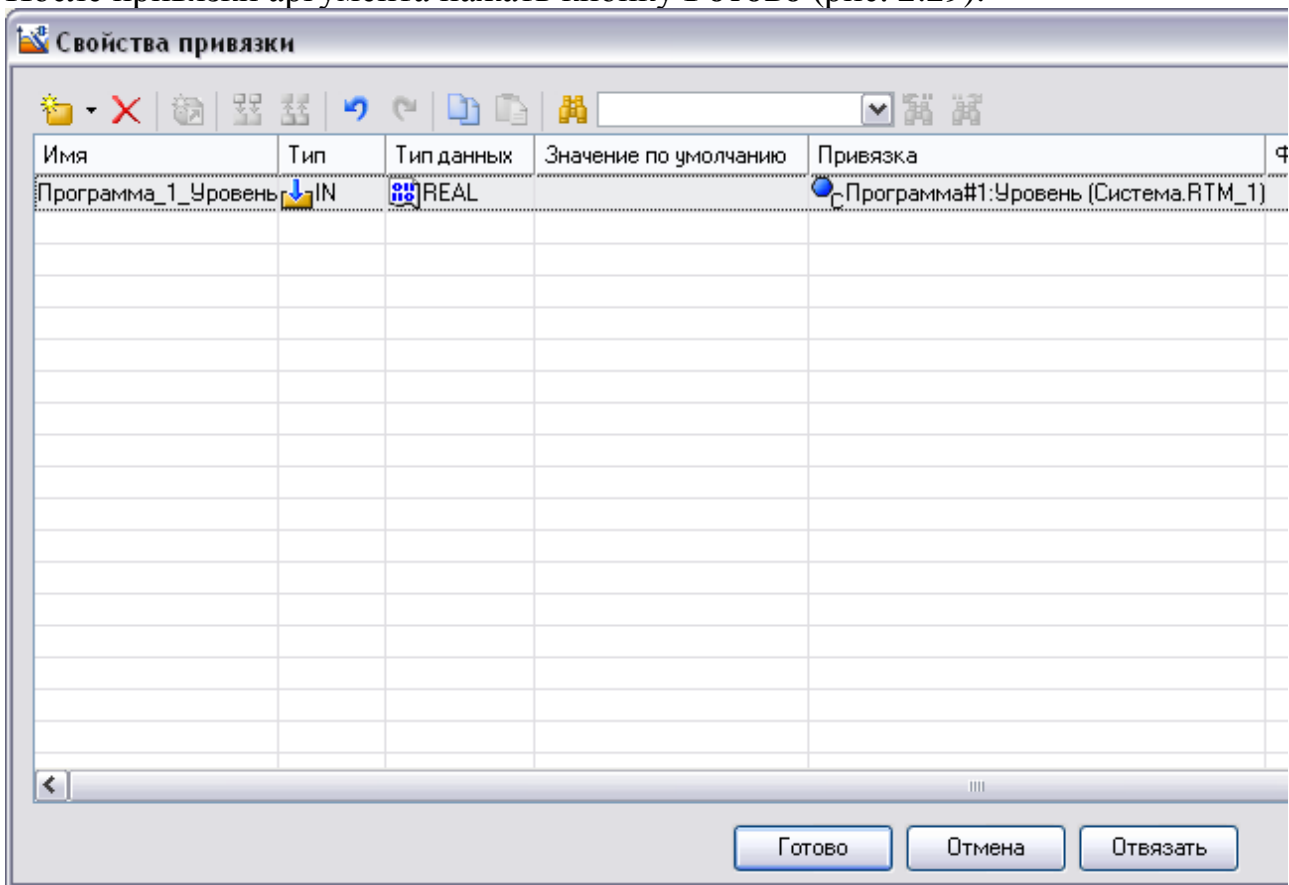


Рисунок 2.29. Свойство привязки

Затем, в свойстве объекта щелкнуть ЛК по значению свойства **Макс**. Изменить его на 200 и нажать **Enter** (рис. 2.30).

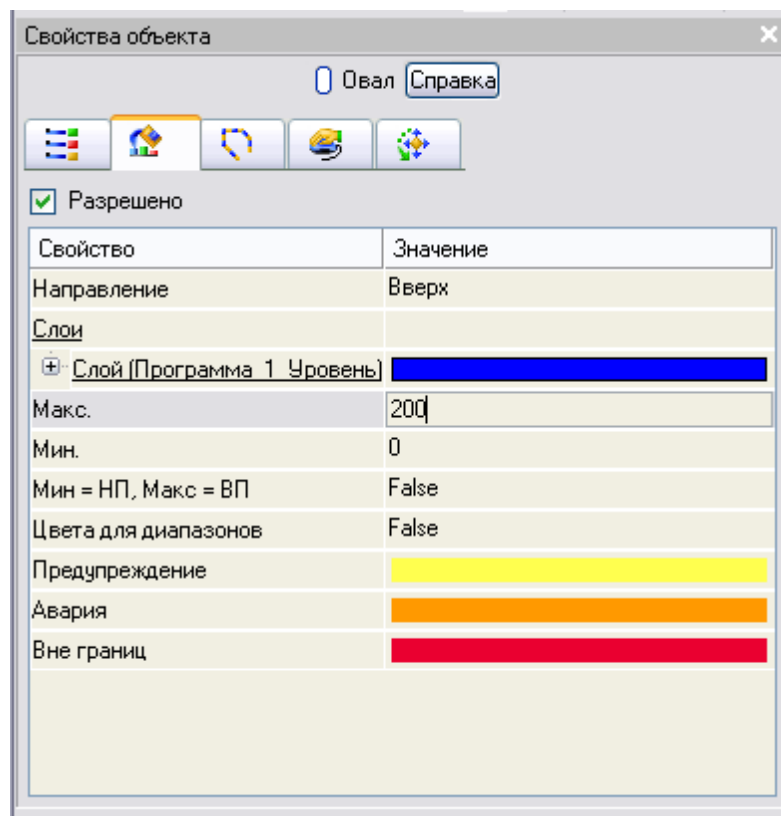


Рисунок 2.30. Изменение свойства **Макс.**




На панели инструментов графического редактора двойным щелчком ЛК по кнопке  вызвать панель **Объемные фигуры** (рис. 2.31)



Рисунок 2.31. Панель инструментов графического редактора – Объемные фигуры - Труба

На данной панели выбрать щелчком ЛК графический элемент (ГЭ) *Труба* , затем, в поле графического редактора, ЛК указать левый верхний угол ГЭ, движением мыши растянуть до необходимого размера и зафиксировать правый нижний угол ЛК.

Если сделанный ГЭ оказался неподходящего размера, надо перейти в режим редактирования, выделив ЛК иконку  на панели инструментов, выделить его и изменить размер ГЭ с помощью мыши.

С помощью данного ГЭ надо создать систему трубопровода (рис. 2.32).

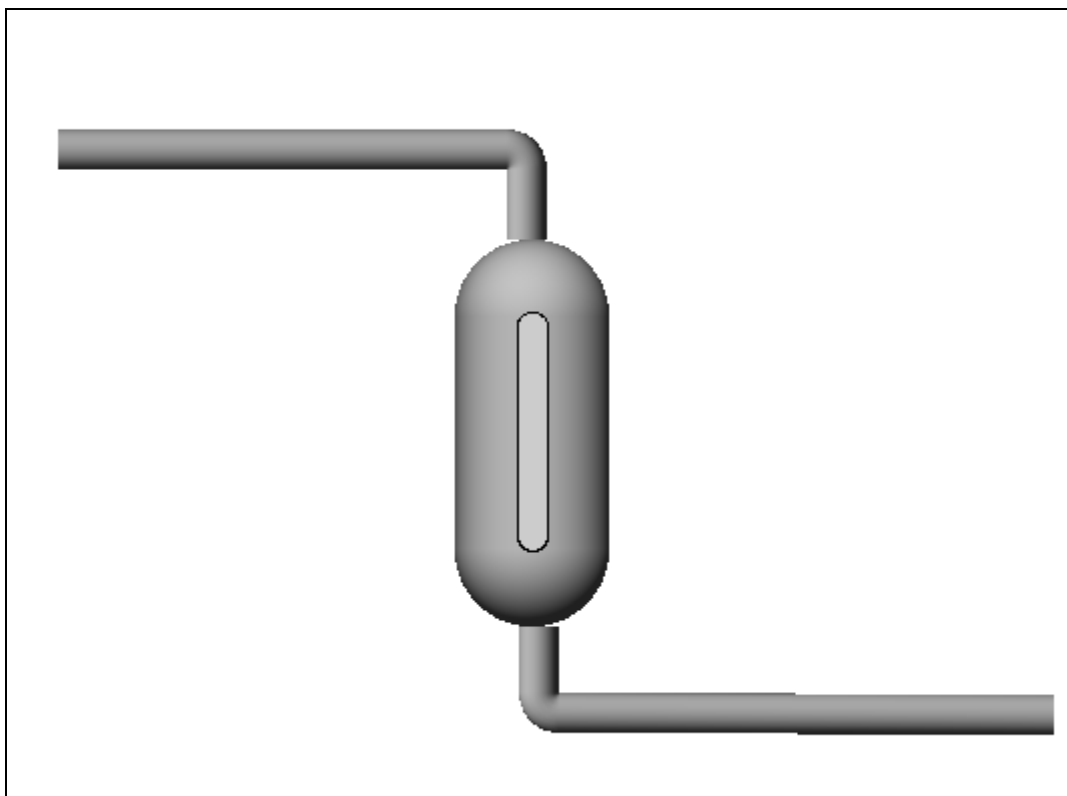




Рисунок 2.32. Графическое изображение ёмкости с трубопроводом  
 На панели инструментов графического редактора двойным щелчком ЛК по кнопке  вызвать панель **Объемные фигуры** (рис. 2.33).



Рисунок 2.33. Панель инструментов графического редактора – Объемные фигуры - Клапан

На данной панели выбрать щелчком ЛК графический элемент (ГЭ) *Клапан* - , затем, в поле графического редактора, ЛК указать левый верхний угол ГЭ, движением мыши растянуть до необходимого размера и зафиксировать правый нижний угол ЛК (рис. 2.34).

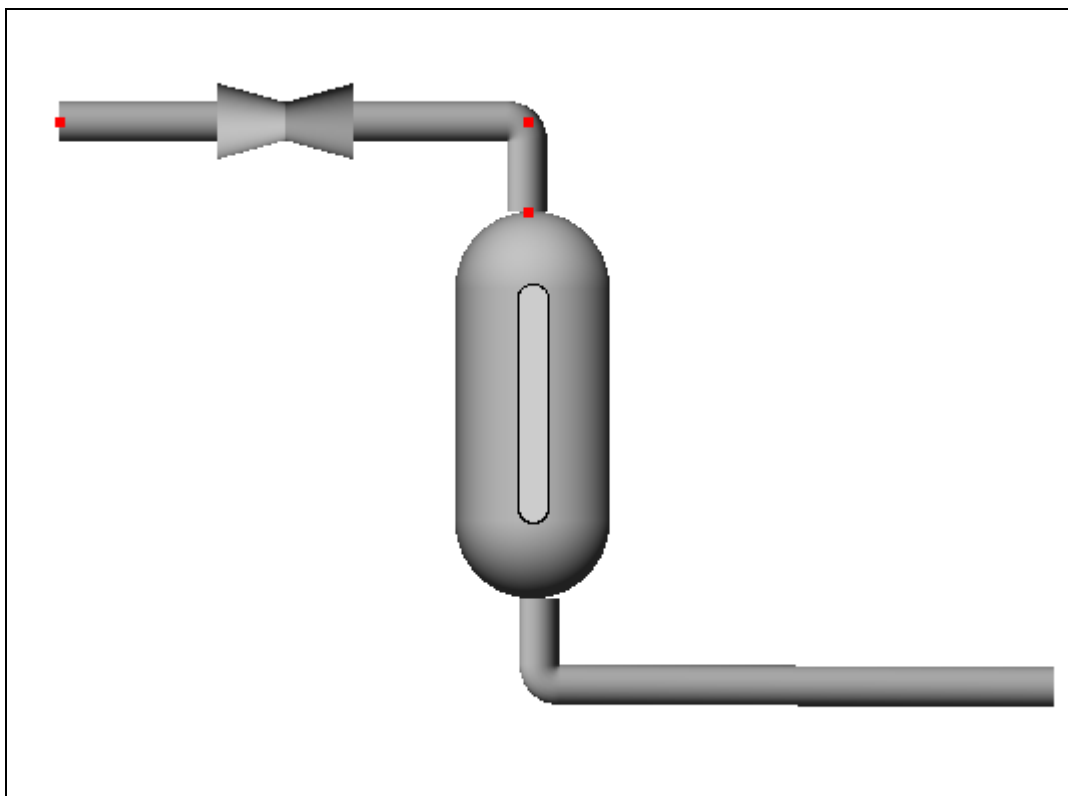



Рисунок 2.34. Графическое изображение ёмкости с трубопроводом и верхним клапаном

Затем перейти в режим редактирования, выделив ЛК иконку  на панели инструментов и двойным щелчком ЛК по ГЭ *Клапан* открыть его свойства (рис. 2.35)

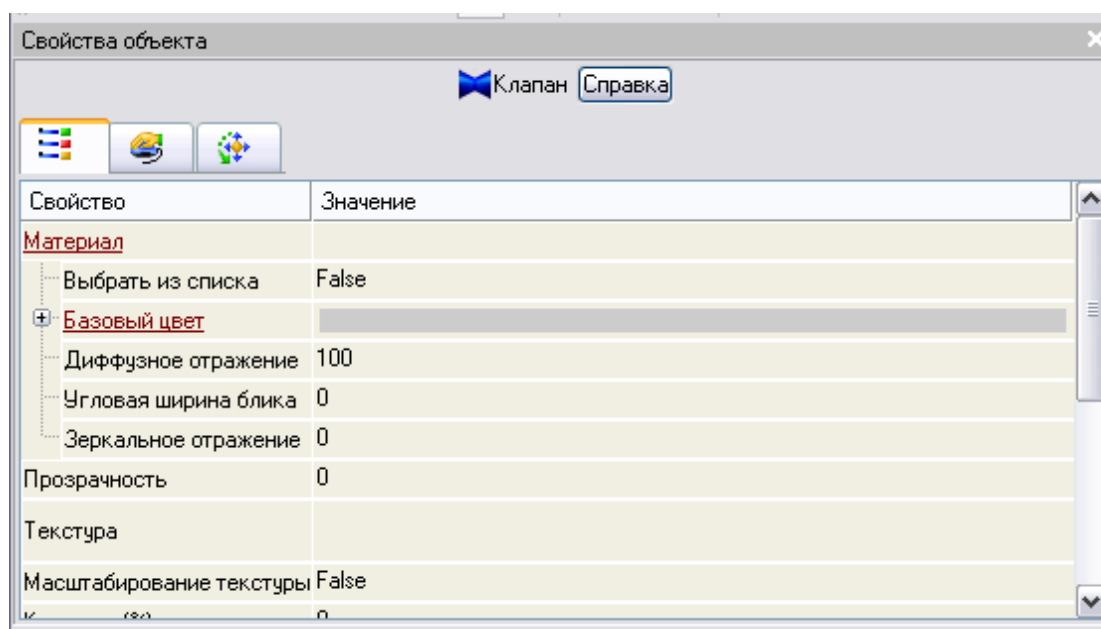


Рисунок 2.35. Свойства объекта *Клапан*

ЛК мыши развернуть меню **Базовый цвет**, затем щелчком ЛК на значении строки **Вид индикации** (в правом поле строки) вызвать список доступных типов динамизации (рис. 2.36).

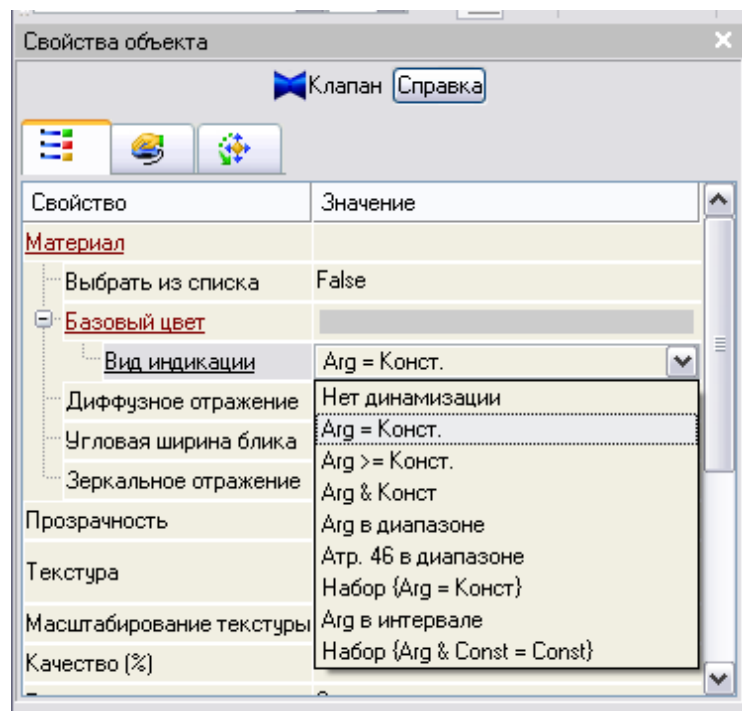


Рисунок 2.36. Список типов динамизации ГЭ Клапан  
 Выбрать тип **Arg = Констант** и открывшемся меню настройки параметров динамизации (рис. 2.37)

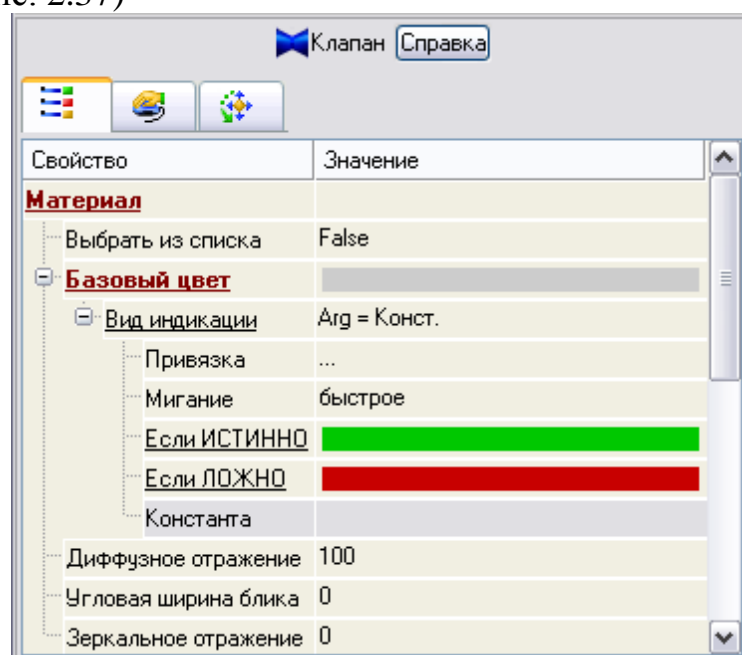


Рисунок 2.37. Установка Вид индикации - Arg = Констант ГЭ Клапан  
 щелчком ЛК в значении поля **Привязка**, вызвать меню Свойства привязки (рис. 2.38)

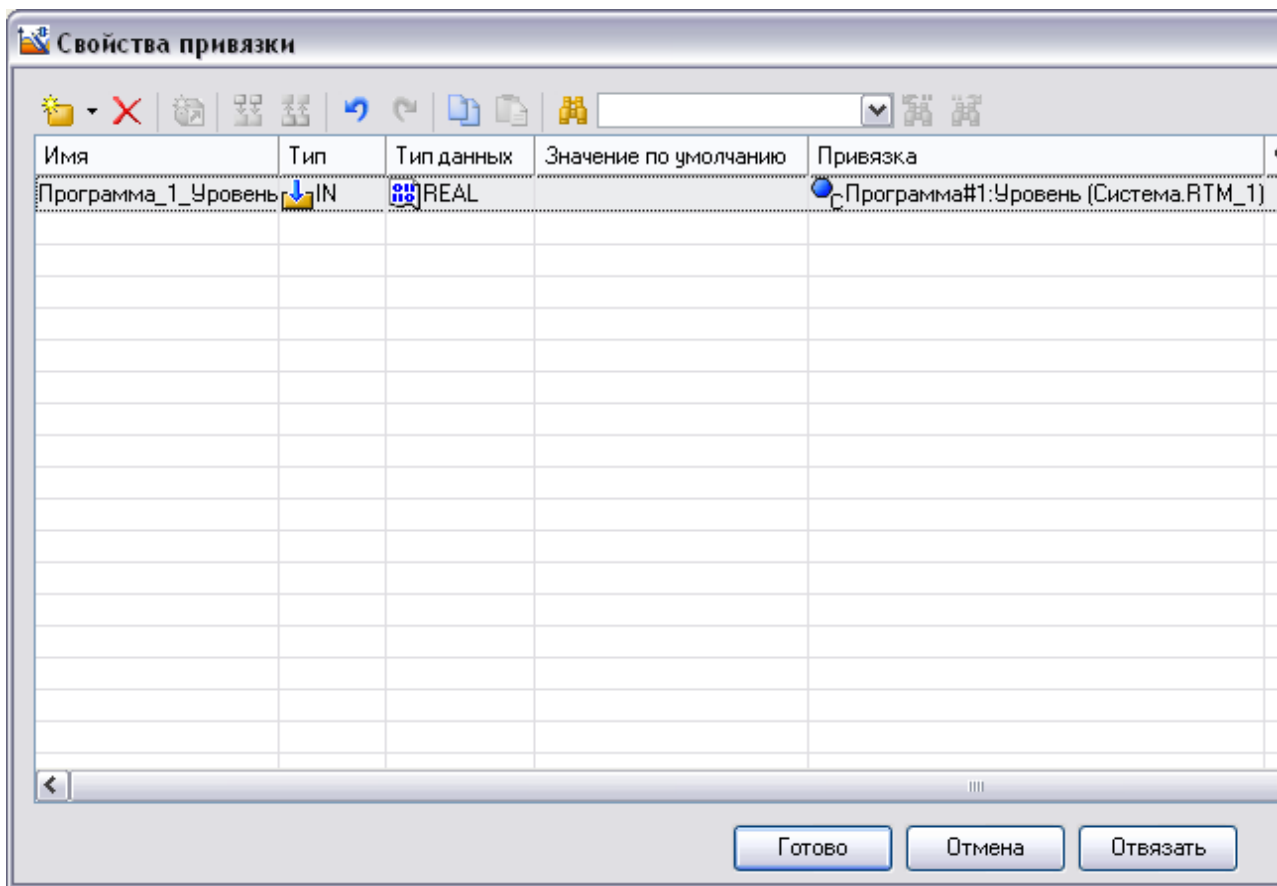



Рисунок 2.38. Свойства привязки

Щелчком ЛК по кнопке  создать новый аргумент (рис. 2.39).

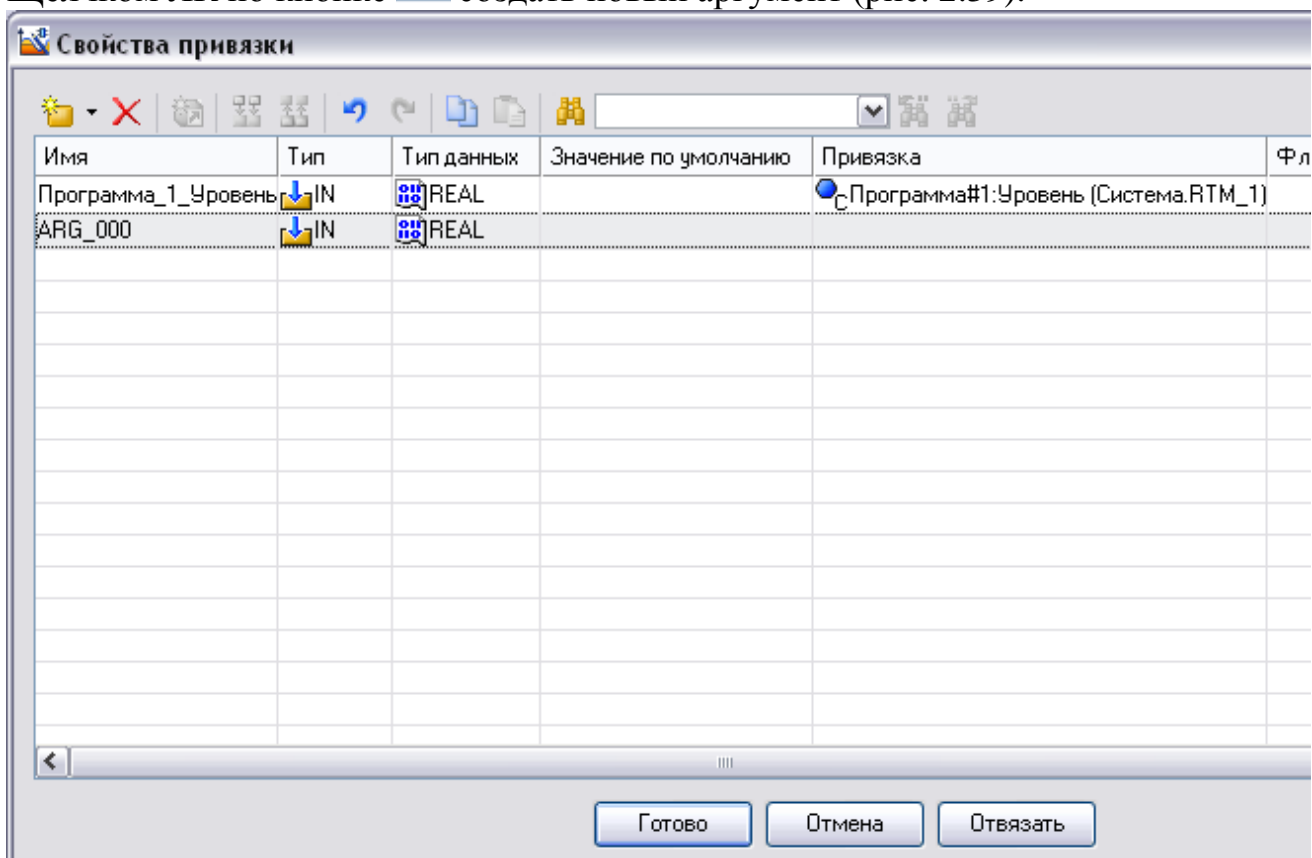



Рисунок 2.39. Свойства привязки – создание нового аргумента

Двойным щелчком ЛК зайти в ячейку *Имя* аргумента и изменить его изменить на **Залив** (введя имя с клавиатуры, завершив ввод нажатием клавиши **Enter**)



[illegible]

Для задания расхода заливаемой жидкости надо на инструментальной панели графического редактора выбрать ЛК иконку ГЭ кнопки . С помощью мыши разместить его под ГЭ Клапан (рис. 2.41).

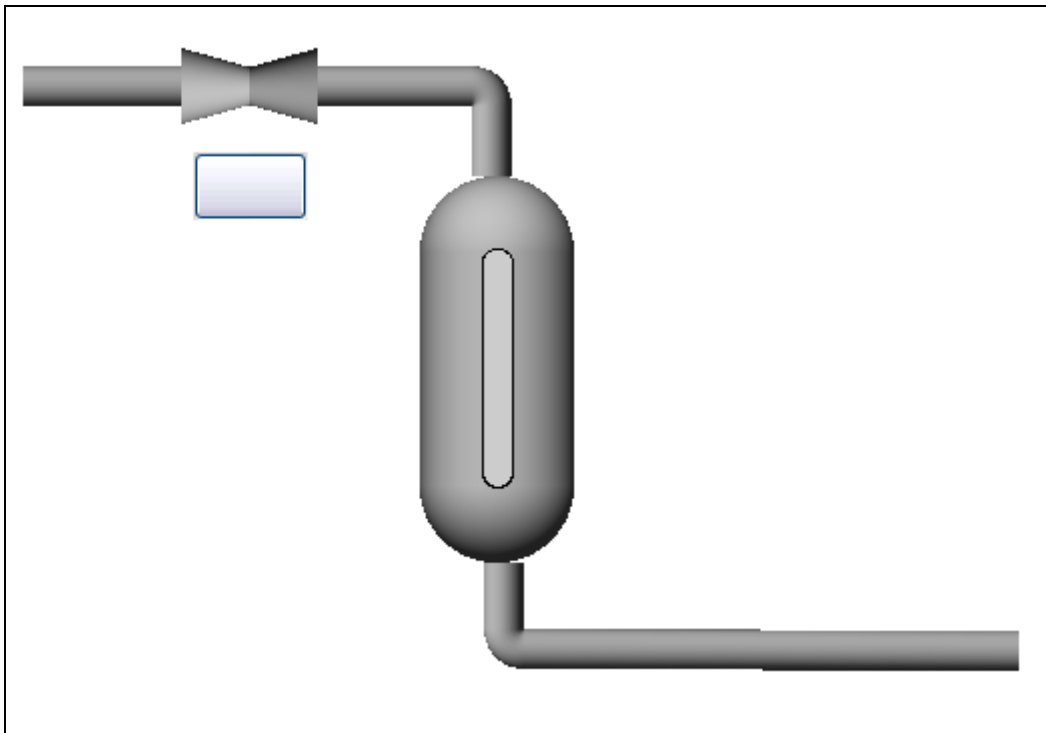



Рисунок 2.41. Создание графического элемента *Кнопка*

Перейти в режим редактирования, выделив ЛК иконку  на панели инструментов.

Щелчком ЛК по ГЭ **Кнопка** вызвать окно его свойств (рис. 2.42)

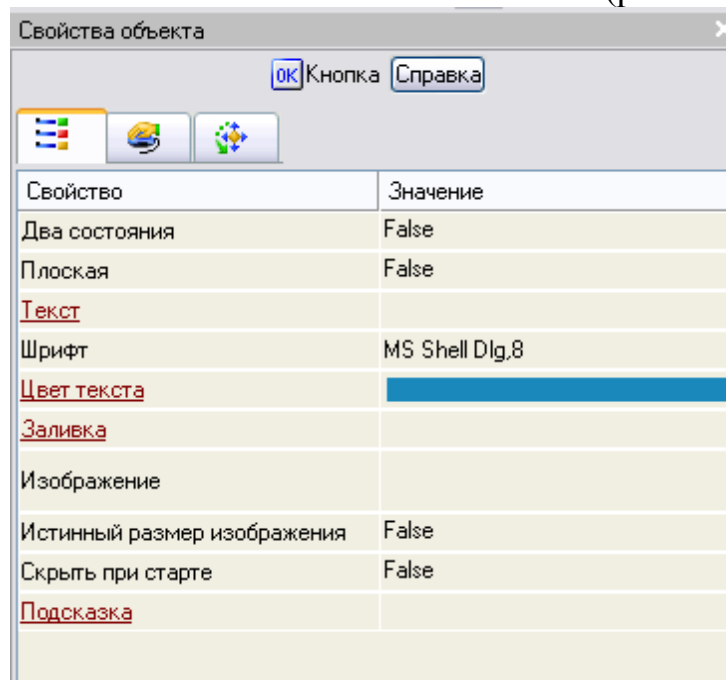


Рисунок 2.42. Свойство графического элемента *Кнопка*

Двойным щелчком ЛК на строке **Текст** вызвать меню **Вид индикации** (рис. 2.43).

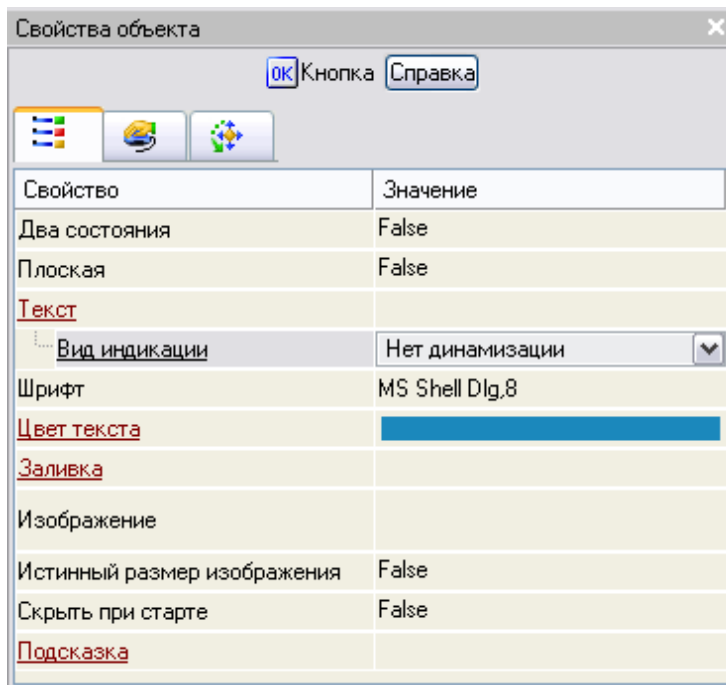


Рисунок 2.43. Вид индикации ГЭ *Кнопка*

Щелчком ЛК по значению меню **Вид индикации** вызвать список доступных типов динамизации атрибута (рис. 2.44)

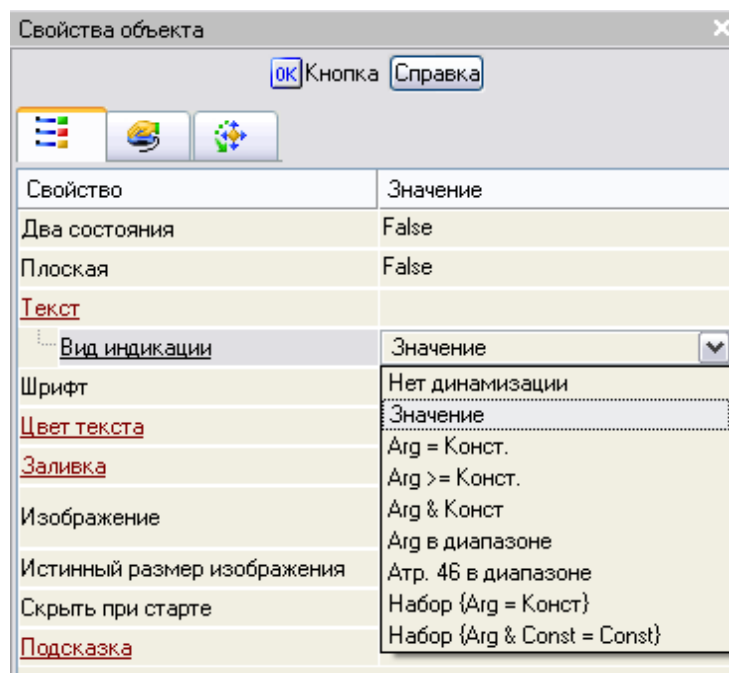


Рисунок 2.44. Список типов динамизации ГЭ *Кнопка*  
С помощью ЛК выбрать тип **Значение** (рис. 2.45).

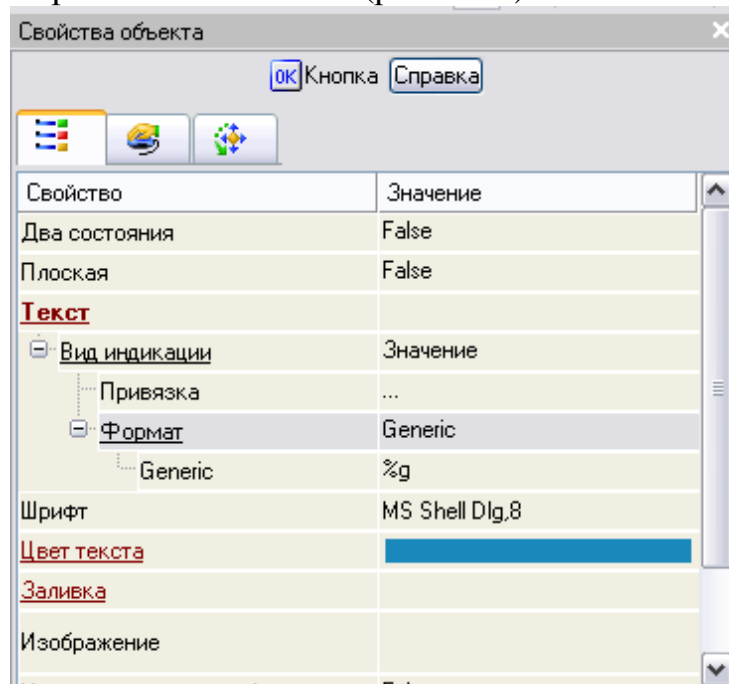


Рисунок 2.45. Установка *Вид индикации* - **Значение** свойства **Текст** ГЭ *Кнопка*  
Щелчком ЛК в значении поля **Привязка**, вызвать меню **Свойства привязки** (рис. 2.46)



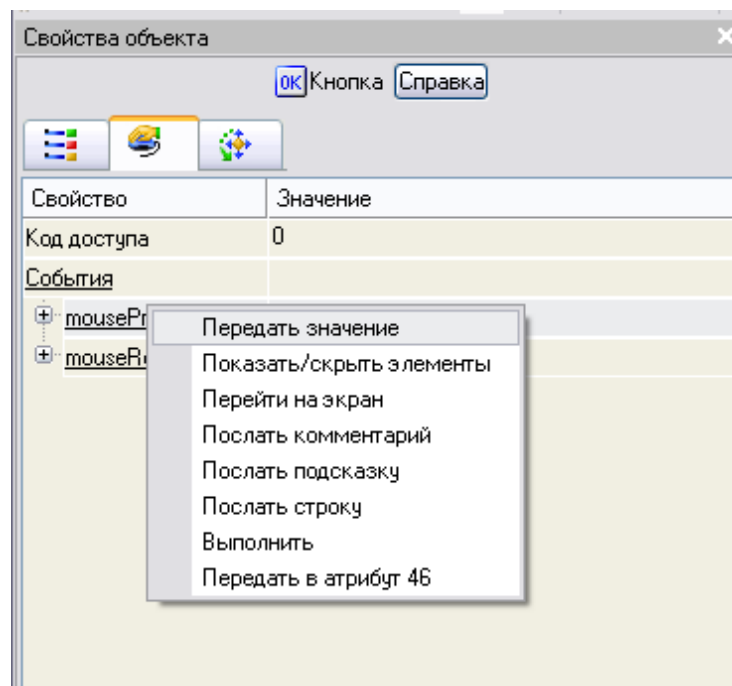


Рисунок 2.48. Создание события **mousePressed** ГЭ *Кнопка* в раскрывшемся меню настроек выбранной команды в поле **Тип передачи** выбрать из списка **Ввести и передать** (рис. 2.49, 2.50)

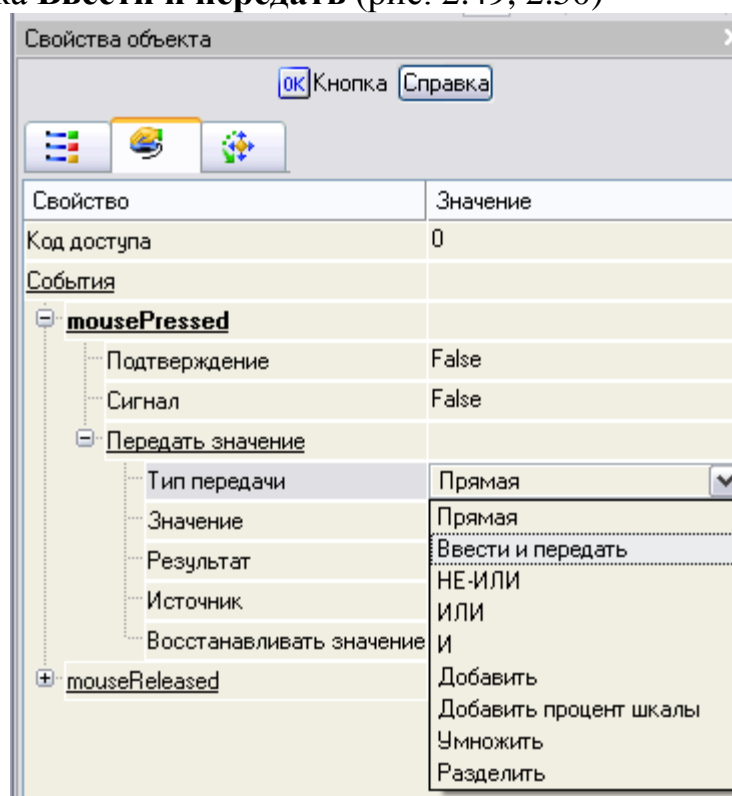


Рисунок 2.49. Список типа передачи- **Ввести и передать** - события **mousePressed** ГЭ *Кнопка*



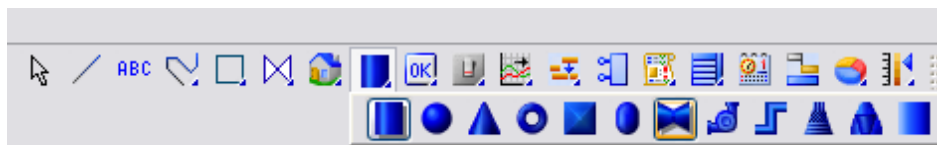



Рисунок 2.52. Панель инструментов графического редактора – Объёмные фигуры - Клапан

На данной панели выбрать щелчком ЛК графический элемент (ГЭ) *Клапан* , затем, в поле графического редактора, ЛК указать левый верхний угол ГЭ, движением мыши растянуть до необходимого размера и зафиксировать правый нижний угол ЛК (рис. 2.53).

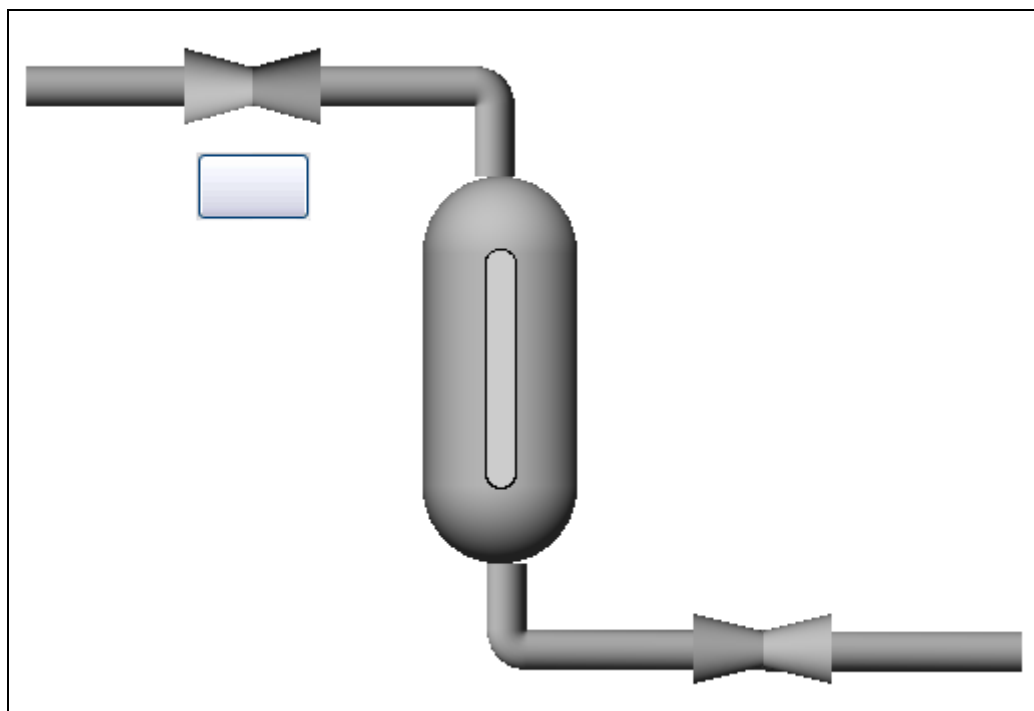



Рисунок 2.53. Графическое изображение ёмкости с трубопроводом и верхним и нижним клапанами

Затем перейти в режим редактирования, выделив ЛК иконку  на панели инструментов и двойным щелчком ЛК по ГЭ *Клапан* открыть его свойства (рис. 2.54).

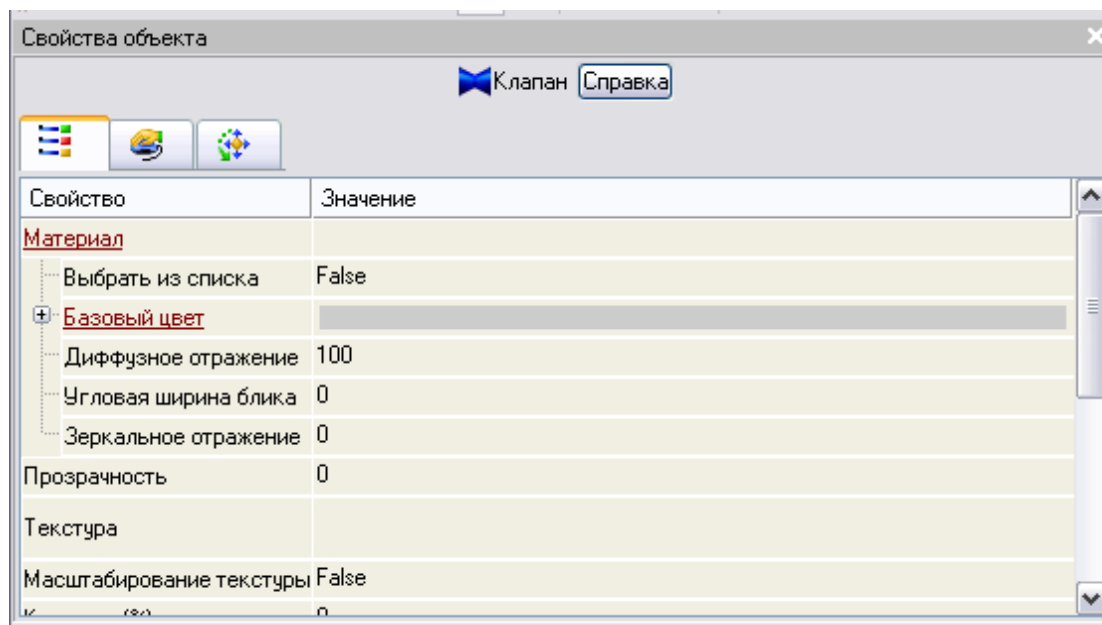


Рисунок 2.54. Свойство графического элемента *Клапан* ЛК мыши развернуть меню **Базовый цвет**, затем щелчком ЛК на значении строки **Вид индикации** (в правом поле строки) вызвать список доступных типов динамизации (рис. 2.55).

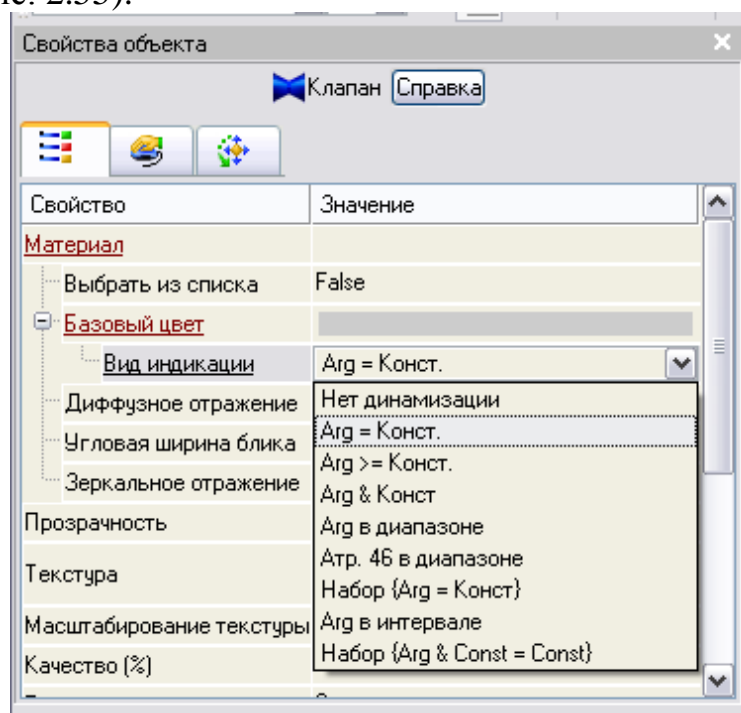


Рисунок 2.55. Список типов динамизации ГЭ *Клапан* Выбрать тип **Arg = Констант** и открывшемся меню настройки параметров динамизации (рис. 2.56).



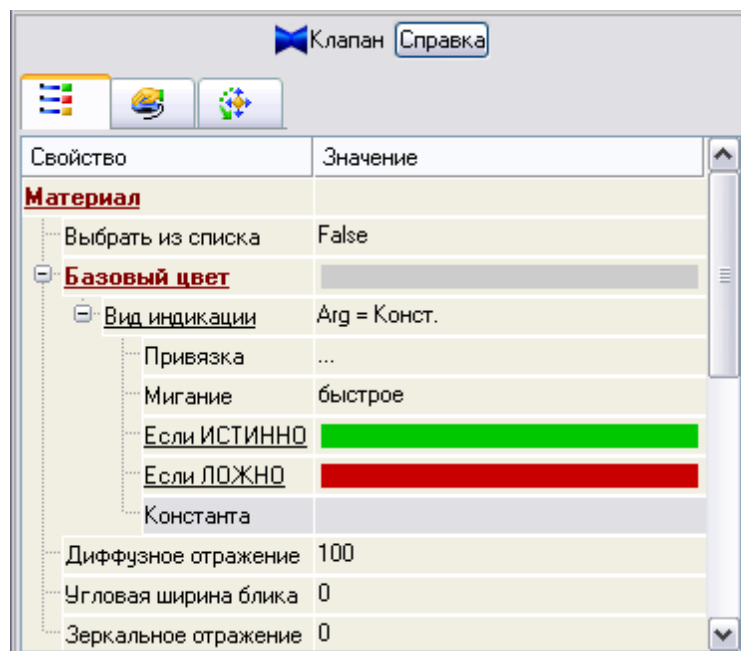


Рисунок 2.56. Установка Вид индикации - Arg = Констант ГЭ Кнопка щелчком ЛК в значении поля **Привязка**, вызвать меню **Свойства привязки** (рис. 2.57)

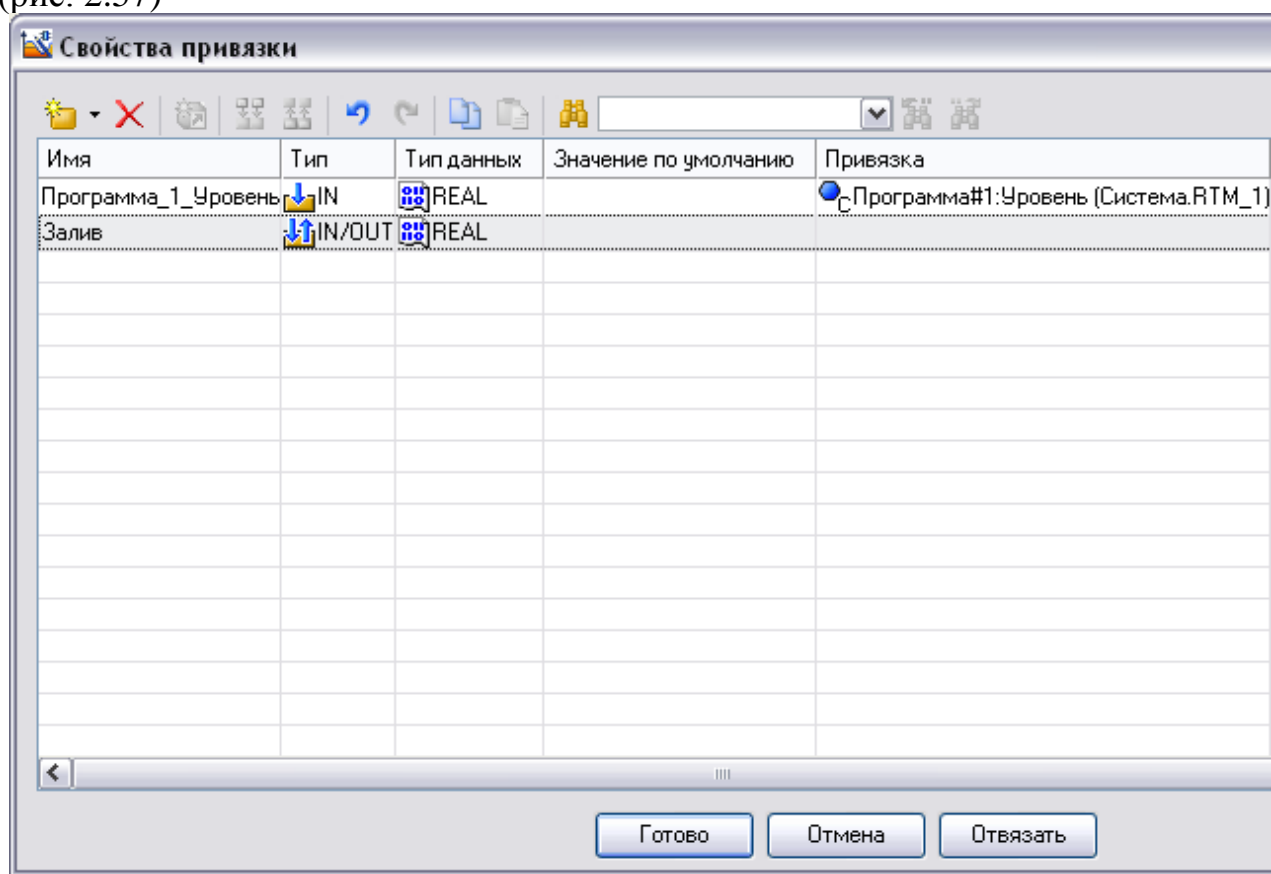


Рисунок 2.57. Свойства привязки

Щелчком ЛК по кнопке  создать новый аргумент (рис. 2.58).

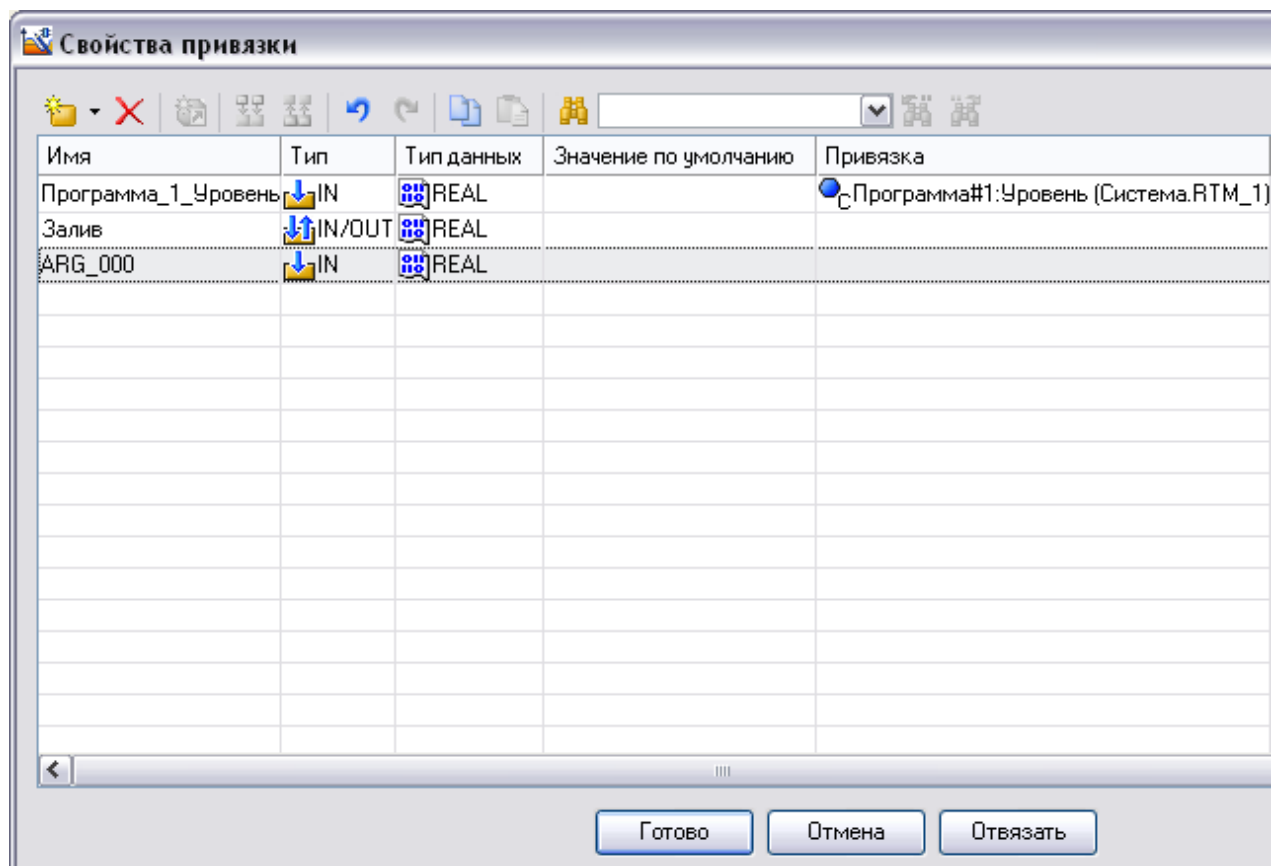



Рисунок 2.58. Свойства привязки – создание аргумента

Двойным щелчком ЛК зайти в ячейку *Имя* аргумента и изменить его изменить на **Слив** (введя имя с клавиатуры, завершив ввод нажатием клавиши **Enter**)

Двойным щелчком ЛК по ячейке *Тип* аргумента вызвать выпадающий список, в котором выбрать тип **IN/OUT**.

Связать ГЭ *Клапан* с данным атрибутом нажав кнопку **Готово**.

Для задания расхода заливаемой жидкости надо на инструментальной панели графического редактора выберем ЛК иконку ГЭ *Кнопки* . С помощью мыши разместить его под ГЭ *Клапан* (рис. 2.59).

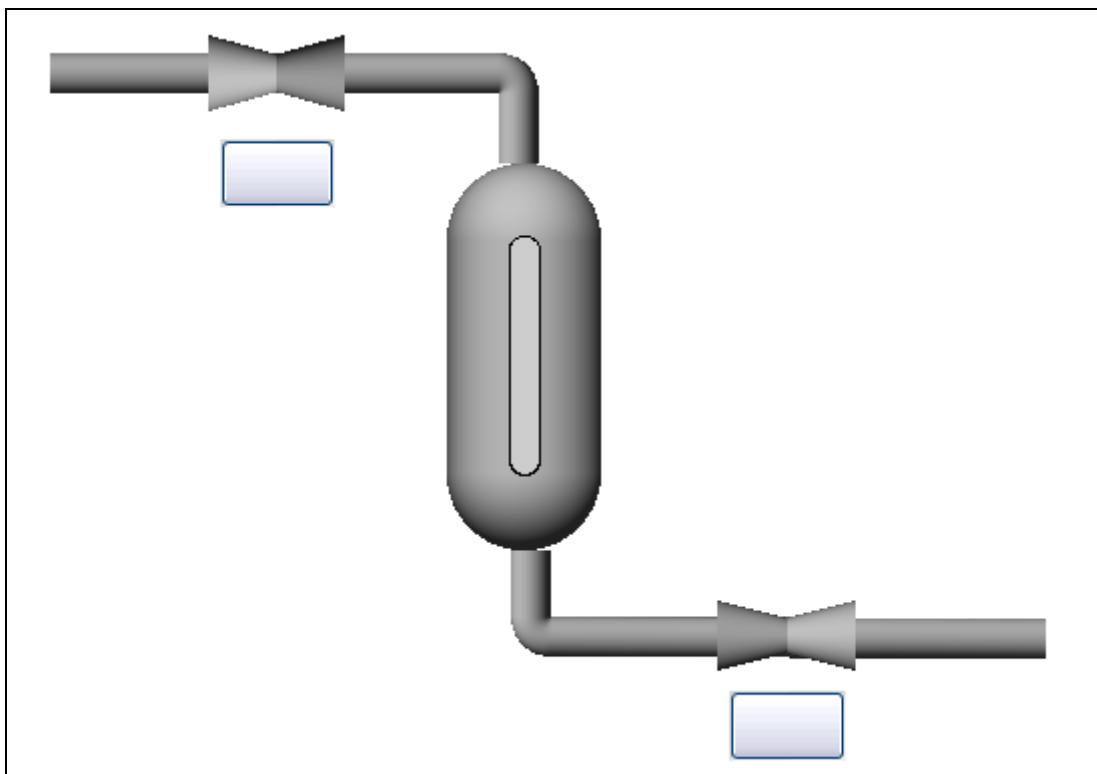



Рисунок 2.59. Графическое изображение ёмкости с трубопроводом – вставка ГЭ  
*Кнопки*

Перейти в режим редактирования, выделив ЛК иконку  на панели инструментов.

Щелчком ЛК по ГЭ **Кнопка** вызвать окно его свойств (рис. 2.60).

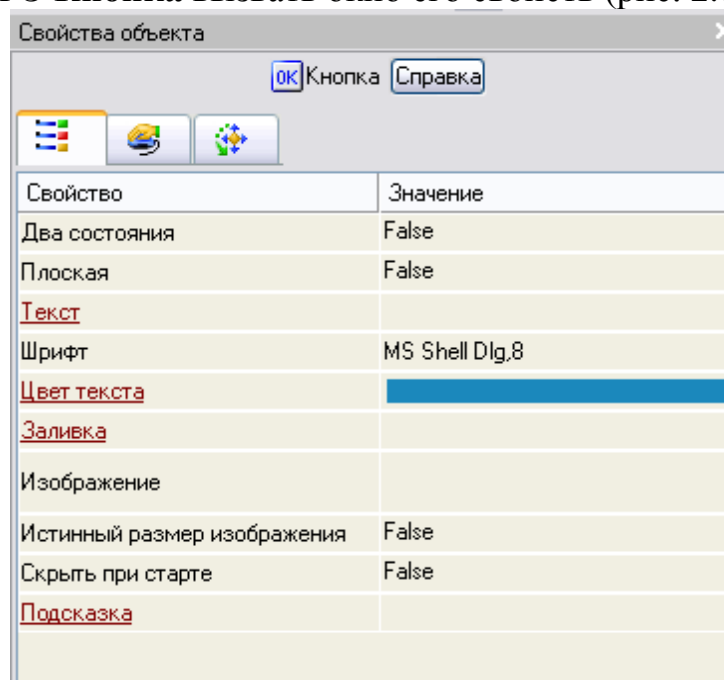


Рисунок 2.60. Свойства ГЭ *Кнопка*

Двойным щелчком ЛК на строке **Текст** вызвать меню **Вид индикации** (рис. 2.61).

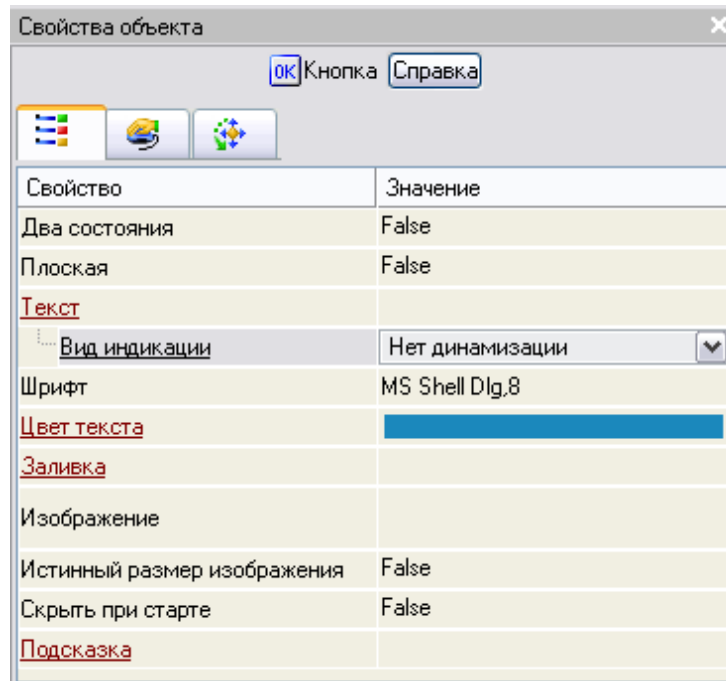


Рисунок 2.61. Вид индикации свойства **Текст** ГЭ *Кнопка*  
Щелчком ЛК по значению меню **Вид индикации** вызвать список доступных типов динамизации атрибута (рис. 2.62)

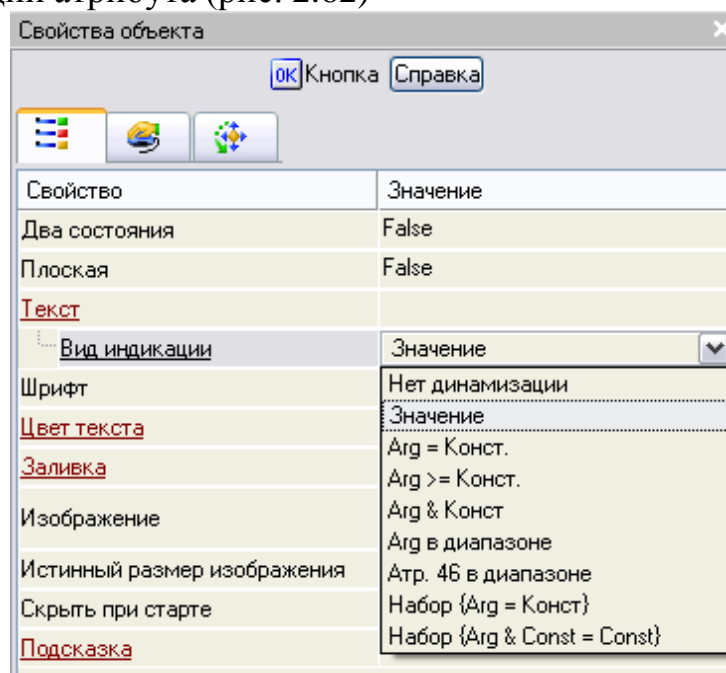


Рисунок 2.62. Список типов динамизации свойства **Текст** ГЭ *Кнопка*  
С помощью ЛК выбрать тип **Значение** (рис. 2.63).

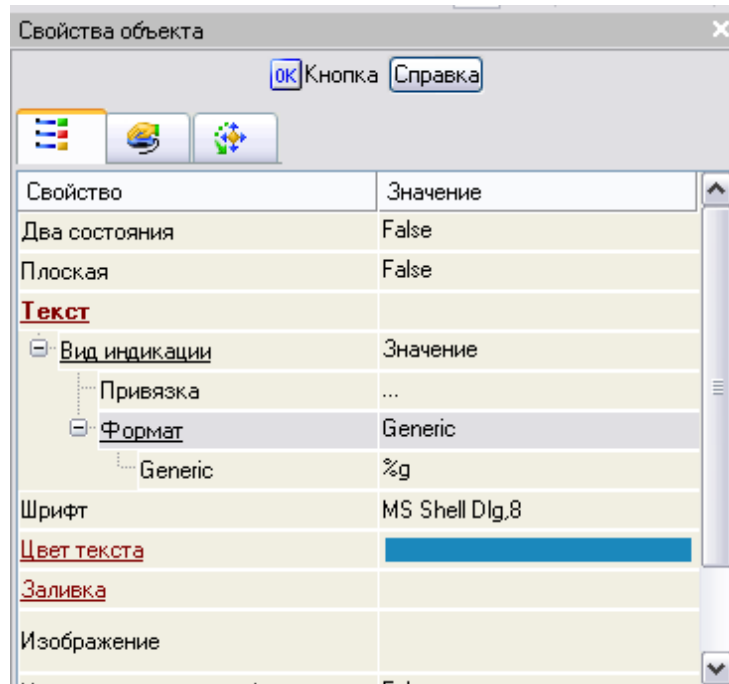


Рисунок 2.63. Установка типа *Значение* свойства **Текст** ГЭ *Кнопка*  
Щелчком ЛК в значении поля **Привязка**, вызвать меню **Свойства привязки**  
(рис. 2.64)

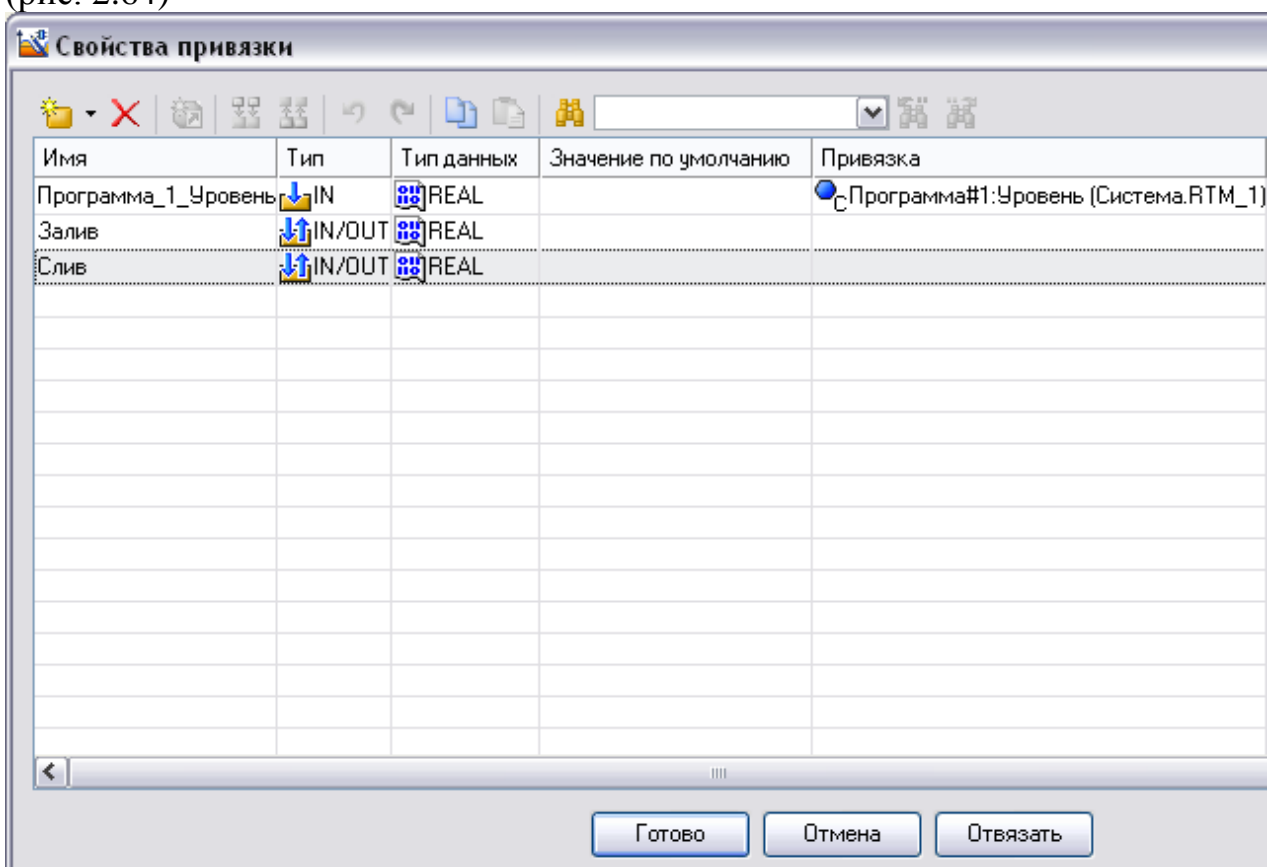


Рисунок 2.64. Свойства привязки  
Выбрать ЛК аргумент **Слив** и связать ГЭ *Кнопка* с данным атрибутом нажав кнопку **Готово**.

Переключиться на бланк **Действия**  свойств ГЭ *Кнопка* (рис. 2.65)

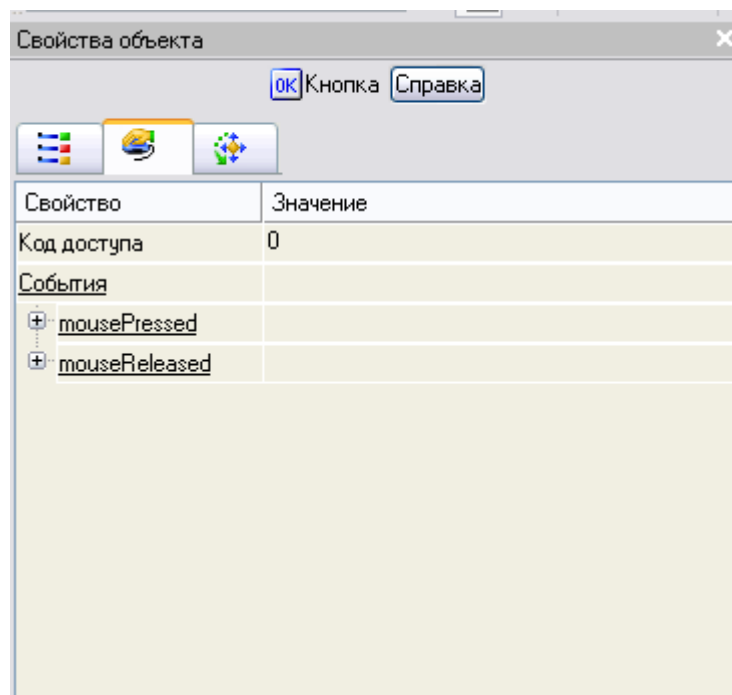


Рисунок 2.65. Бланк Действия свойств ГЭ Кнопка

ПК мыши по событию **mousePressed** вызвать контекстное меню и выбрать из списка ЛК команду **Передать значение** (рис. 2.66)

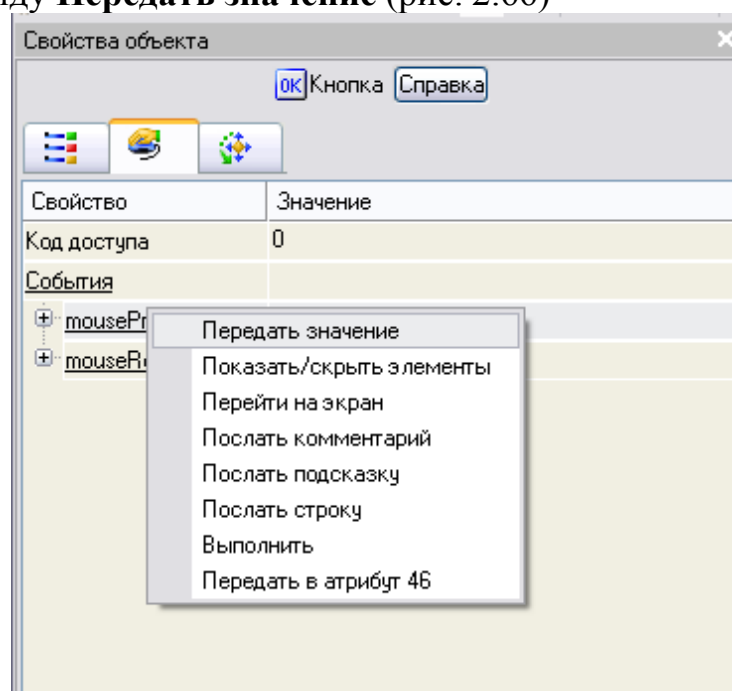


Рисунок 2.66. Создание события **mousePressed** ГЭ Кнопка

в раскрывшемся меню настроек выбранной команды в поле **Тип передачи** выбрать из списка **Ввести и передать** (рис. 2.67, 2.68)

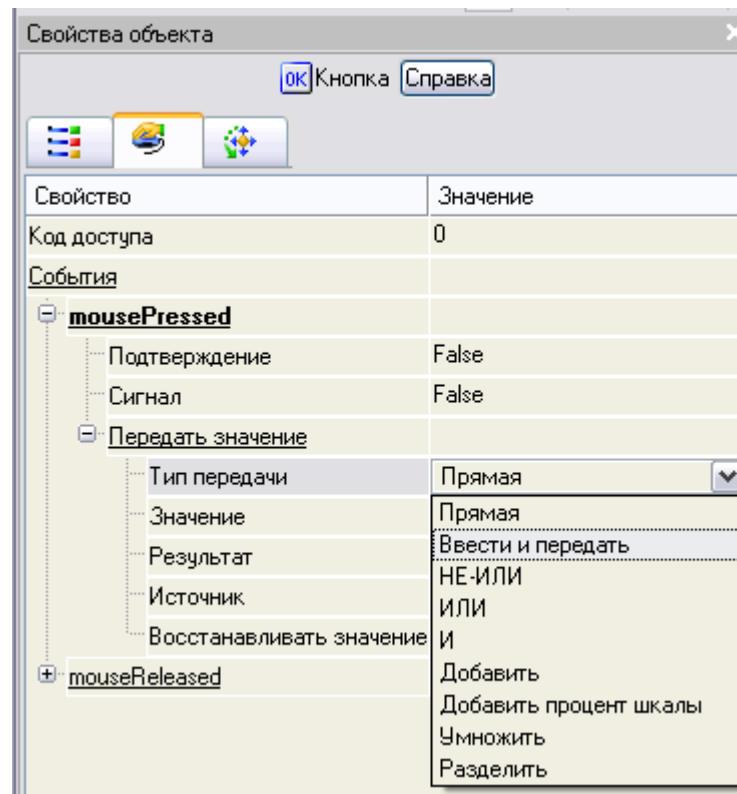


Рисунок 2.67. Список типа передачи- **Ввести и передать** - события **mousePressed** ГЭ *Кнопка*

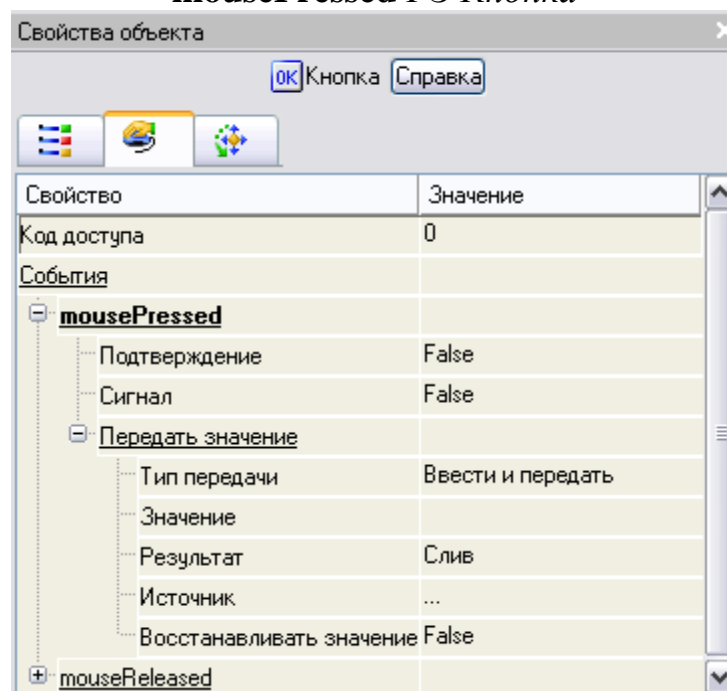
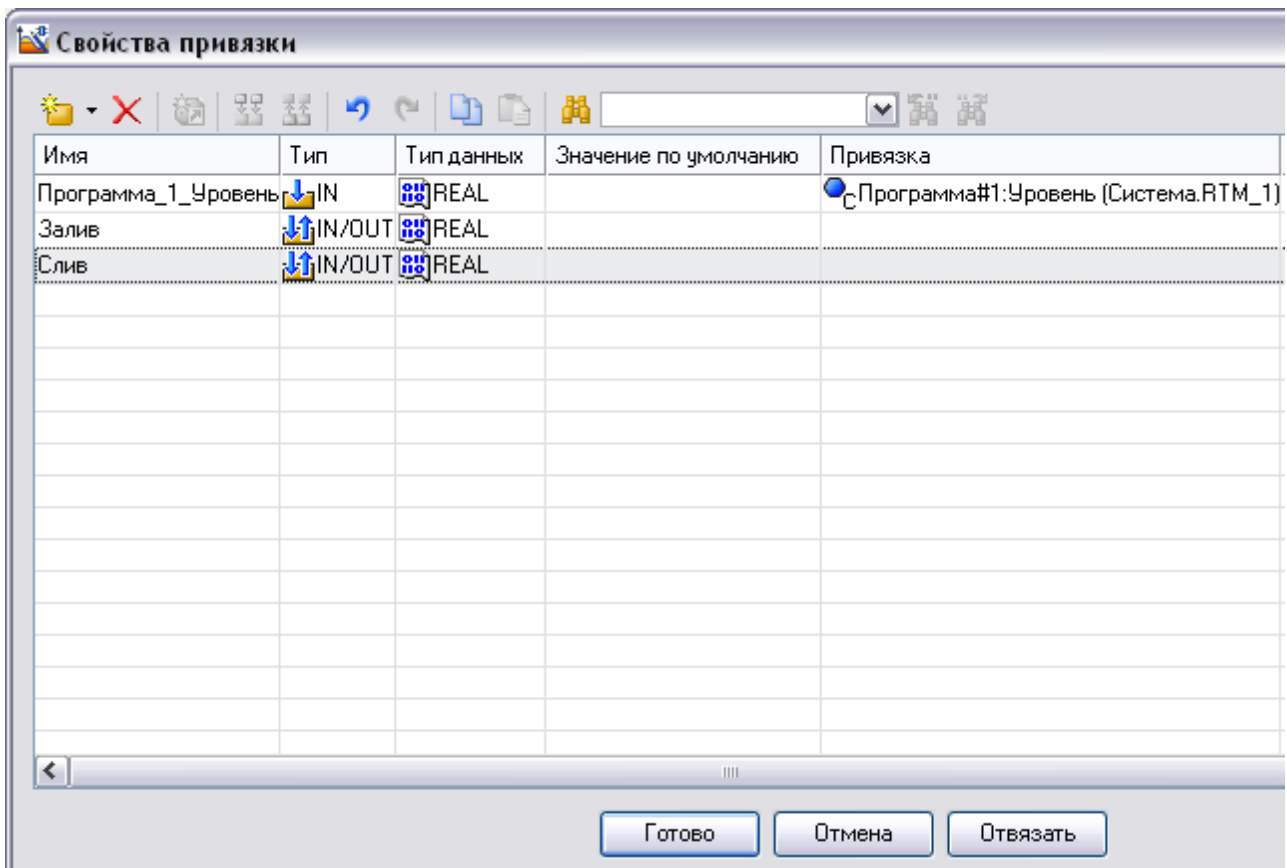



Рисунок 2.68. Установка типа передачи- **Ввести и передать** - события **mousePressed** ГЭ *Кнопка*

щелчком ЛК в поле **Результат** вызвать табличный редактор аргументов (рис. 2.69).



### Рисунок 2.69. Свойства привязки

Выбрать аргумент **Слив** и привязать его к ГЭ кнопкой **Готово**

На панели инструментов графического редактора двойным щелчком ЛК по кнопке  вызвать панель **Приборы** (рис. 2.70)

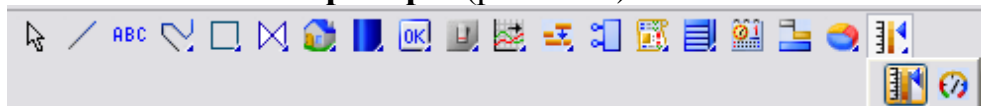



Рисунок 2.70. Панели инструментов графического редактора – панель

## Приборы

На данной панели выбрать щелчком ЛК графический элемент (ГЭ) *Ползунок* , затем, в поле графического редактора, ЛК указать левый верхний угол ГЭ, движением мыши растянуть до необходимого размера и зафиксировать правый нижний угол ЛК (рис. 2.71).



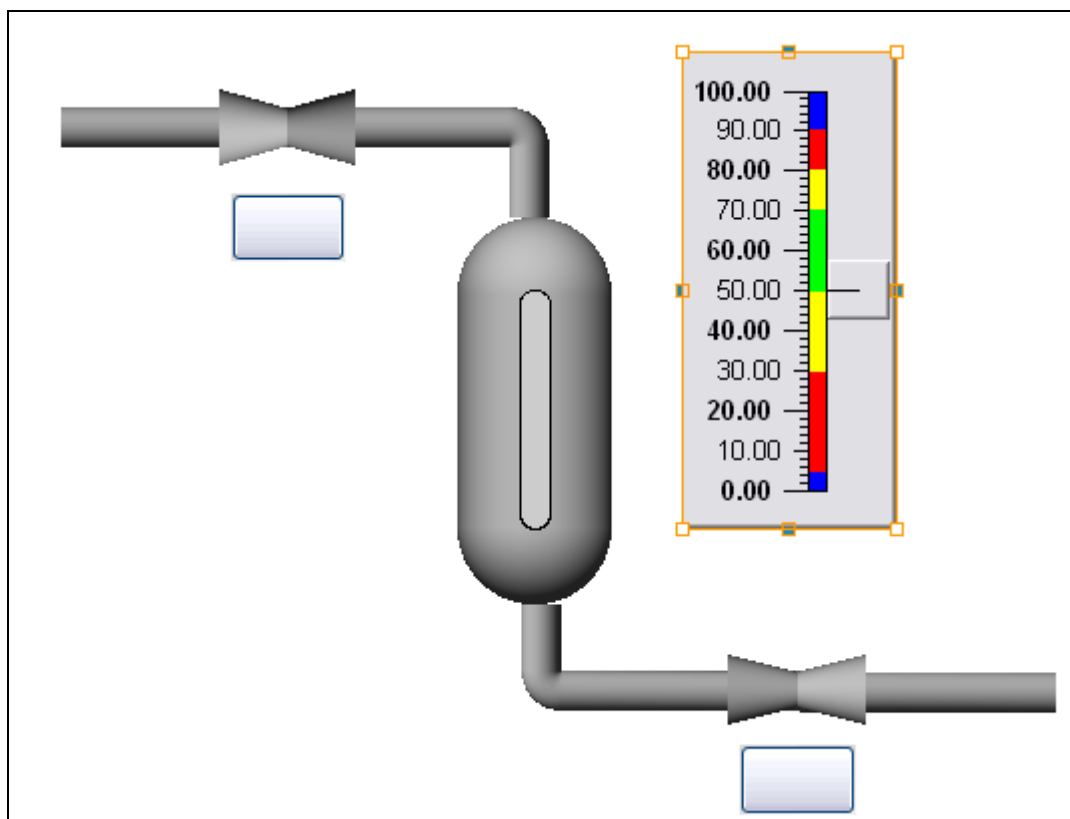



Рисунок 2.71. Вставка ГЭ Ползунок

Затем перейти в режим редактирования, выделив ЛК иконку  на панели инструментов и двойным щелчком ЛК по ГЭ Ползунок открыть его свойства (рис. 2.72)

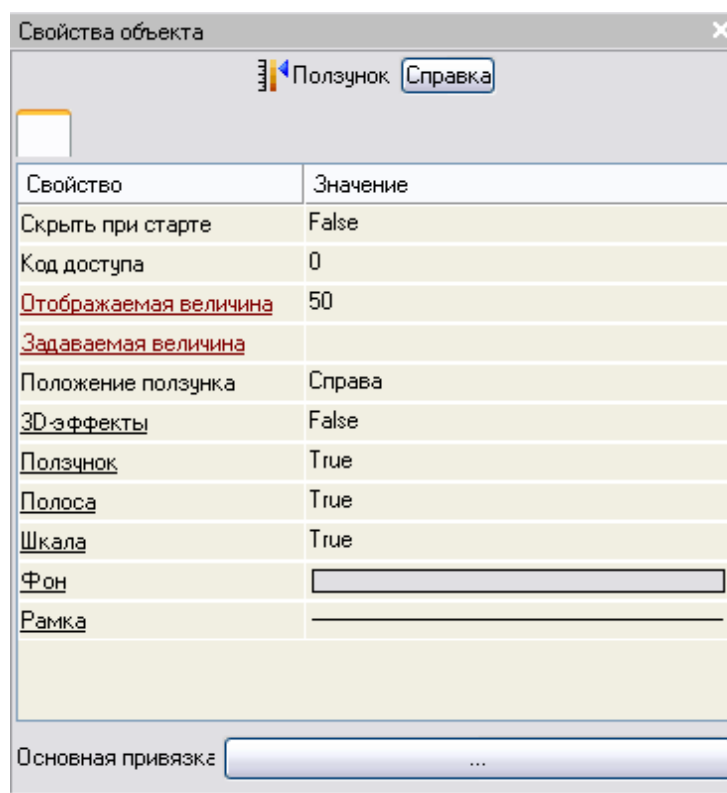


Рисунок 2.72. Свойства ГЭ Ползунок

Двойным щелчком ПК по свойству **Отображаемая величина** открыть меню **Привязка** (рис. 2.73)

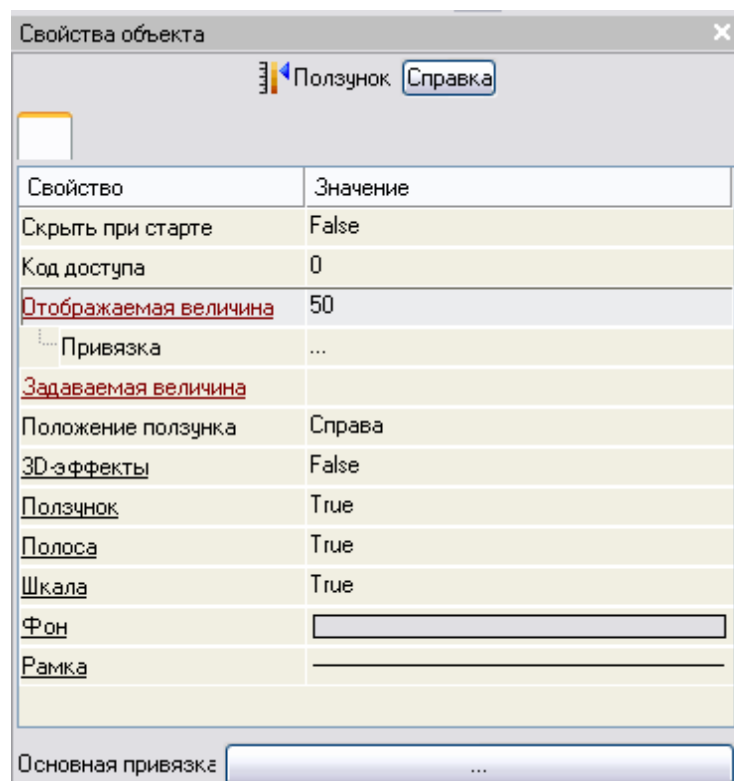


Рисунок 2.73. Свойство **Отображаемая величина** ГЭ *Ползунок*  
Щелчком ЛК по значению меню **Привязка** вызвать табличный редактор аргументов (рис. 2.74).

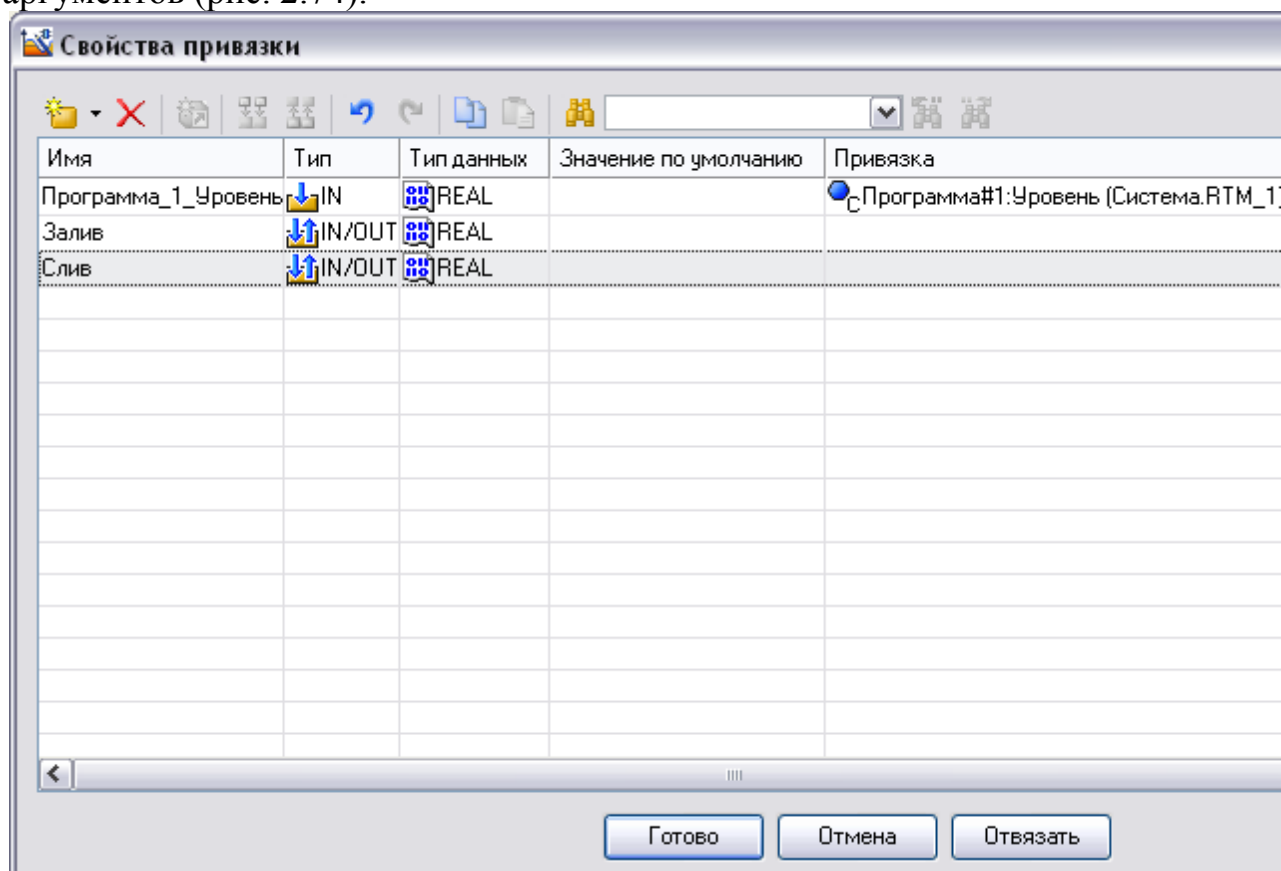



Рисунок 2.74. Свойства привязки  
Щелчком ЛК по кнопке  создать новый аргумент (рис. 2.75)

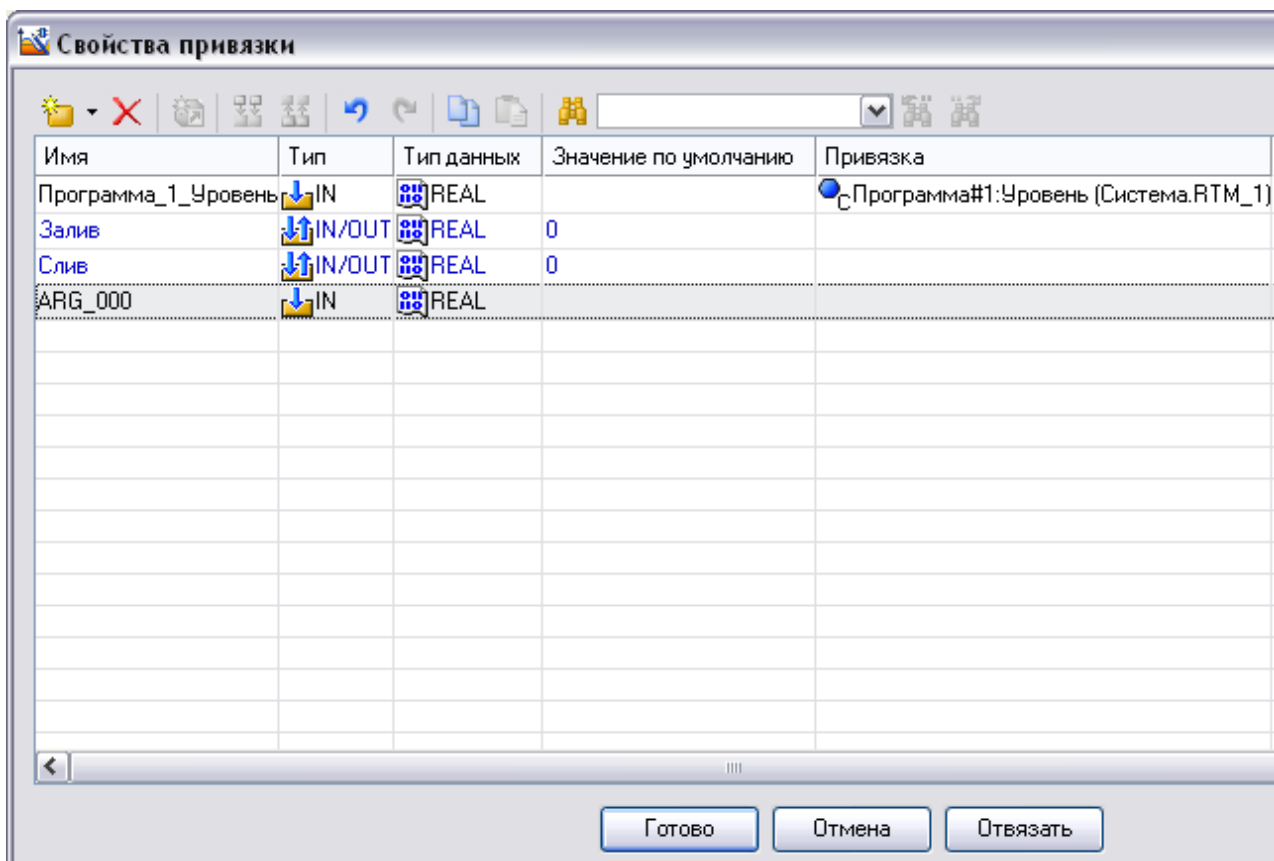


Рисунок 2.75. Свойства привязки – новый аргумент

Двойным щелчком ЛК зайти в ячейку *Имя* аргумента и изменить его изменить на **Задание** (введя имя с клавиатуры, завершив ввод нажатием клавиши **Enter**)  
 Двойным щелчком ЛК по ячейке *Тип* аргумента вызвать выпадающий список, в котором выбрать тип **OUT** (рис. 2.76).

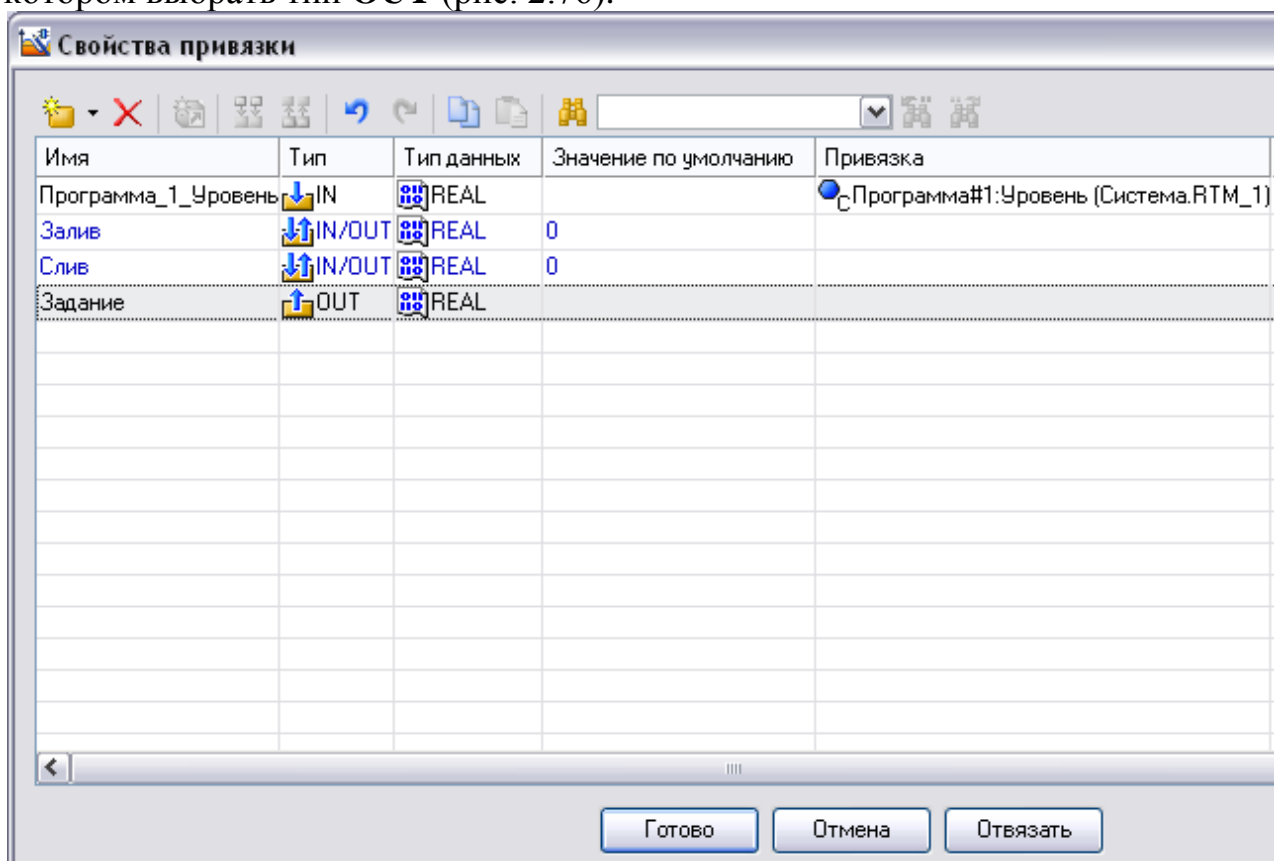


Рисунок 2.76. Свойства привязки – изменение типа аргумента

Связать ГЭ *Ползунок* с данным атрибутом нажав кнопку **Готово**.

Двойным щелчком ПК по свойству **Задаваемая величина** открыть меню **Привязка** (рис. 2.77)

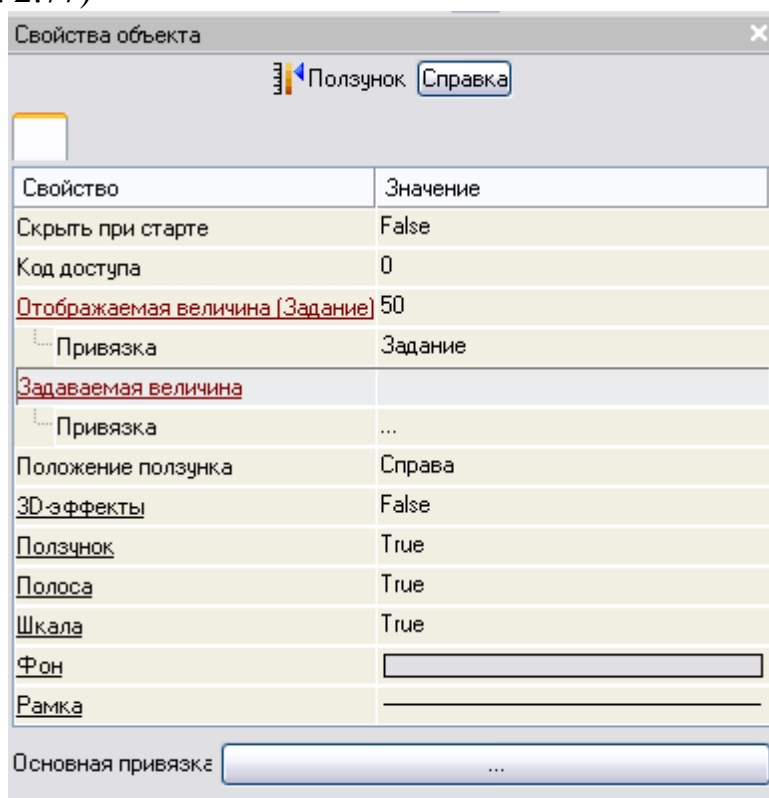


Рисунок 2.77. Свойство **Задаваемая величина** ГЭ *Ползунок*  
Щелчком ЛК по значению меню **Привязка** вызвать табличный редактор аргументов (рис. 2.78)

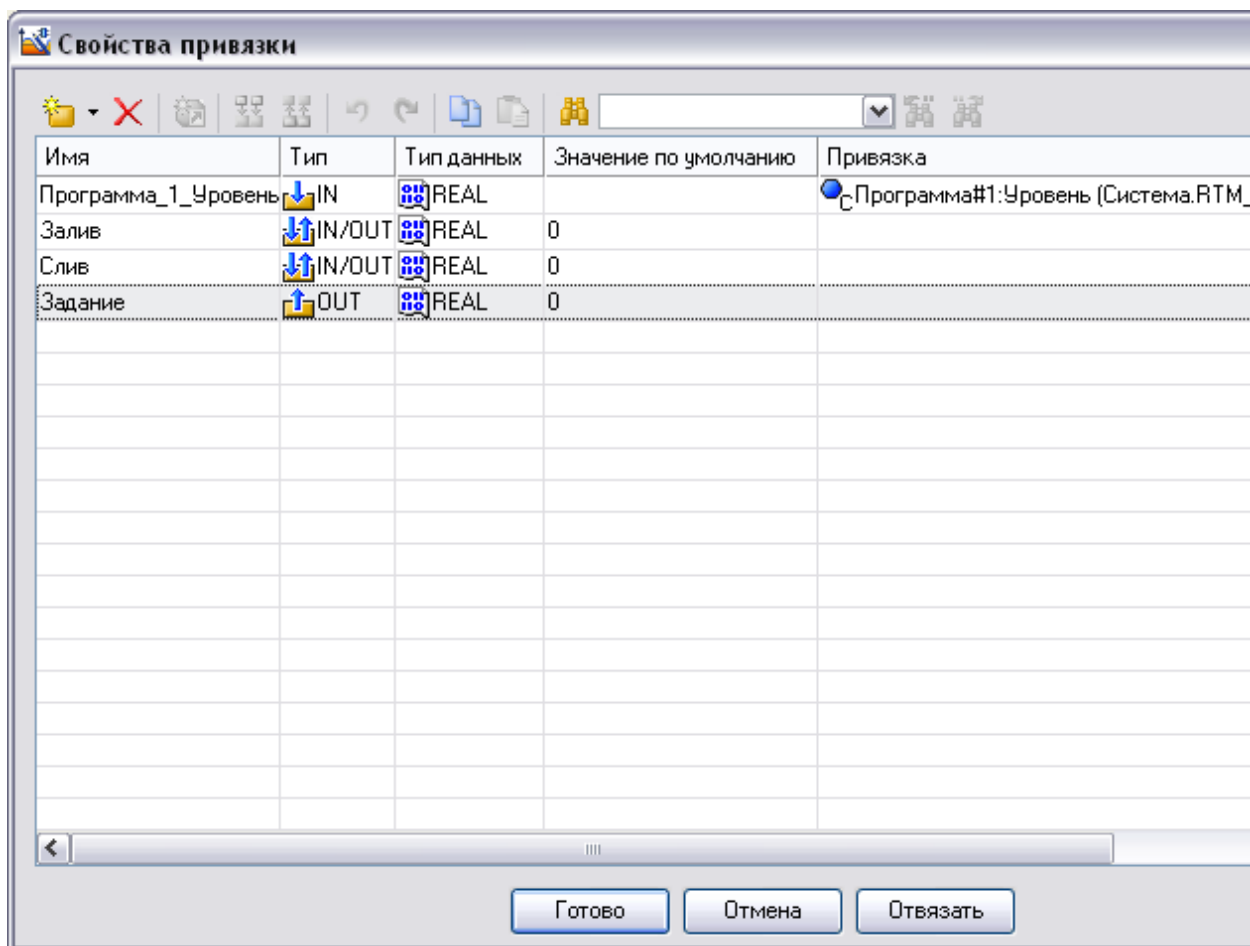


Рисунок 2.78. Свойства привязки

ЛК мыши выбрать аргумент **Задание** и связать его с ГЭ *Ползунок* нажав кнопку **Готово**.

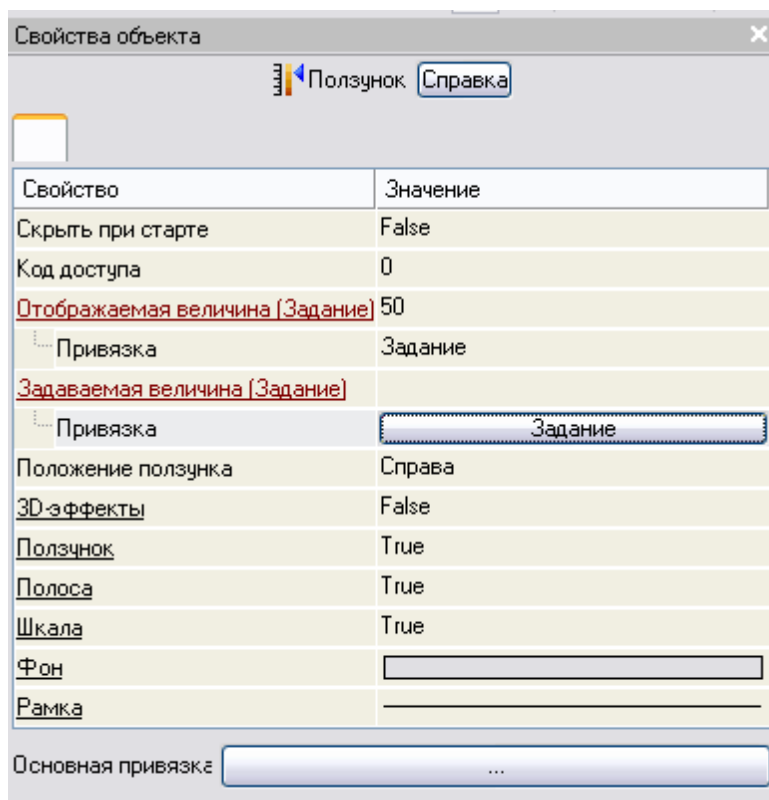


Рисунок 2.79. Свойства ГЭ *Ползунок*

Двойным щелчком по строке **Полоса** вызвать её меню (рис. 2.80)

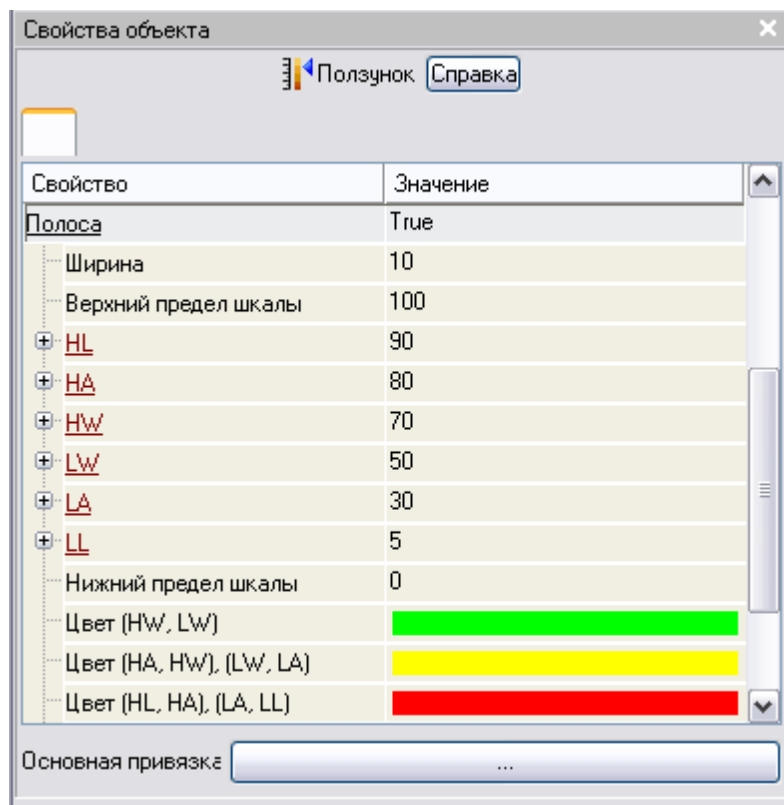


Рисунок 2.80. Свойство **Полоса** ГЭ *Ползунок*

Щелкнув по значению **Верхний предел шкалы** изменить её на 200 и нажать **Enter** (рис. 2.81).

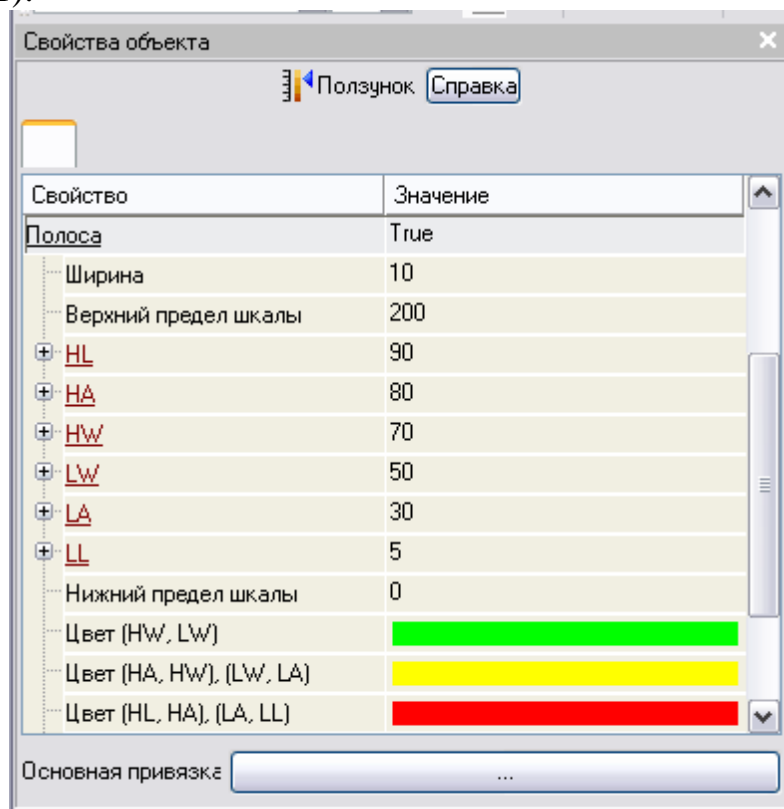



Рисунок 2.81. Изменение **Верхнего предела** свойства **Полоса** ГЭ *Ползунок*

Для точного задания уровня жидкости на инструментальной панели графического редактора выбрать ЛК иконку ГЭ кнопки . С помощью мыши разместить его под ГЭ *Ползунок* (рис. 2.82).

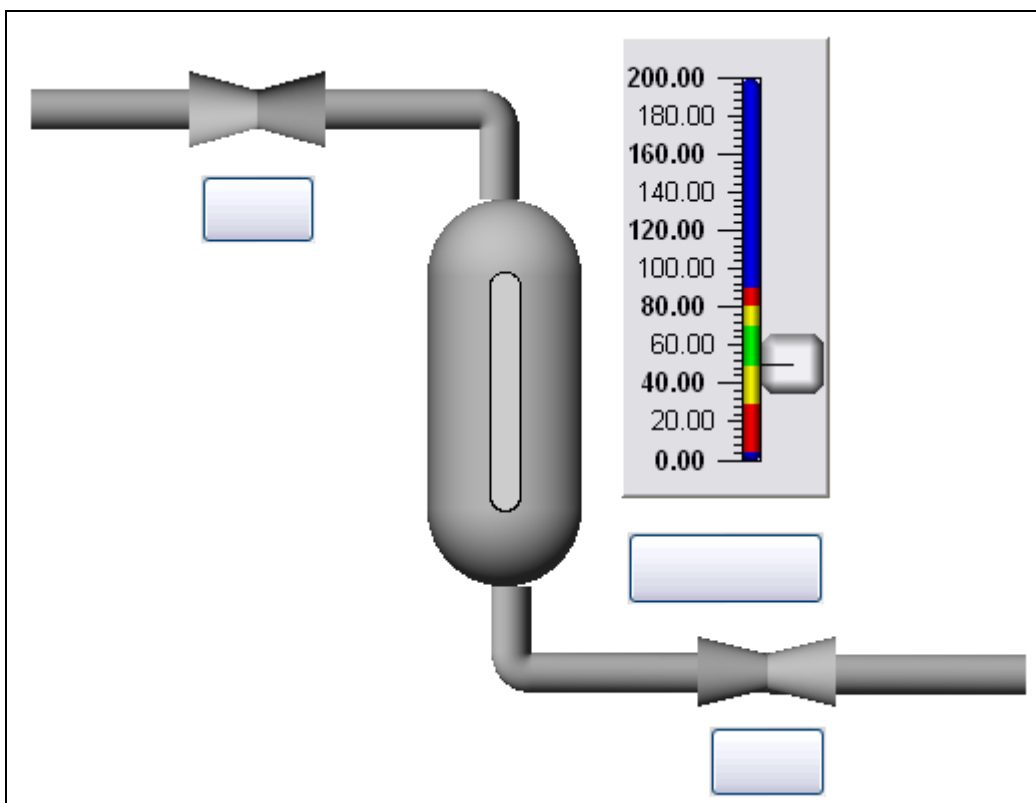



Рисунок 2.82. Размещение ГЭ кнопки

Перейти в режим редактирования, выделив ЛК иконку  на панели инструментов.

Щелчком ЛК по ГЭ **Кнопка** вызвать окно его свойств (рис. 2.83)

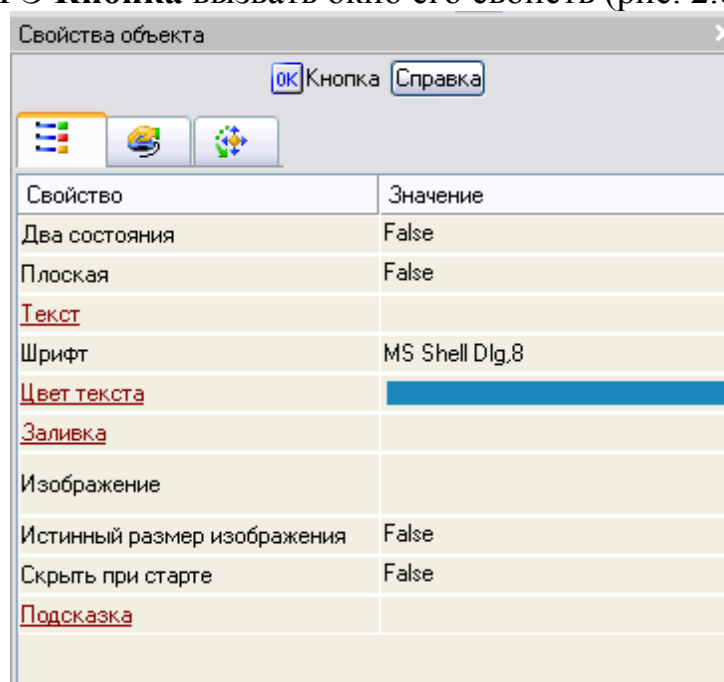


Рисунок 2.83. Свойства ГЭ кнопки

Двойным щелчком ЛК на строке **Текст** вызвать меню **Вид индикации** (рис. 2.84).

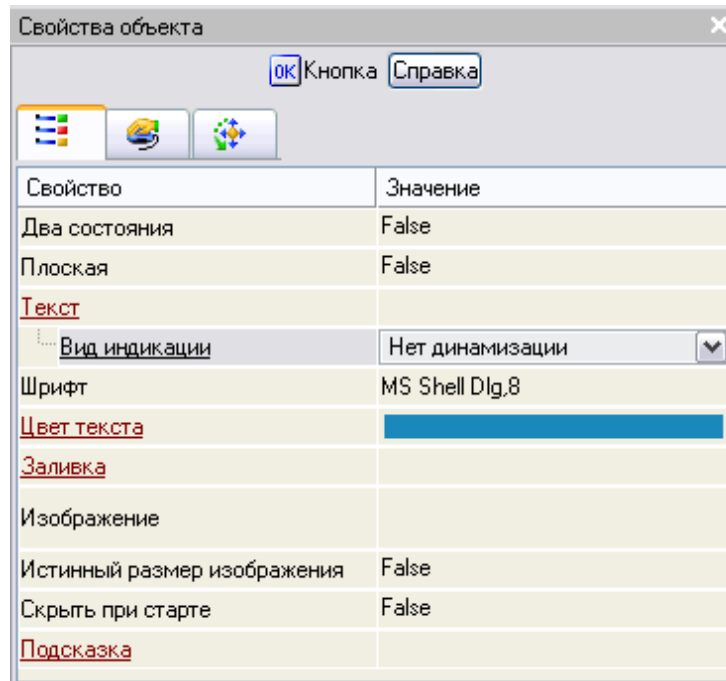


Рисунок 2.84. Свойство **Вид индикации** ГЭ *кнопка*

Щелчком ЛК по значению меню **Вид индикации** вызвать список доступных типов динамизации атрибута (рис. 2.85).

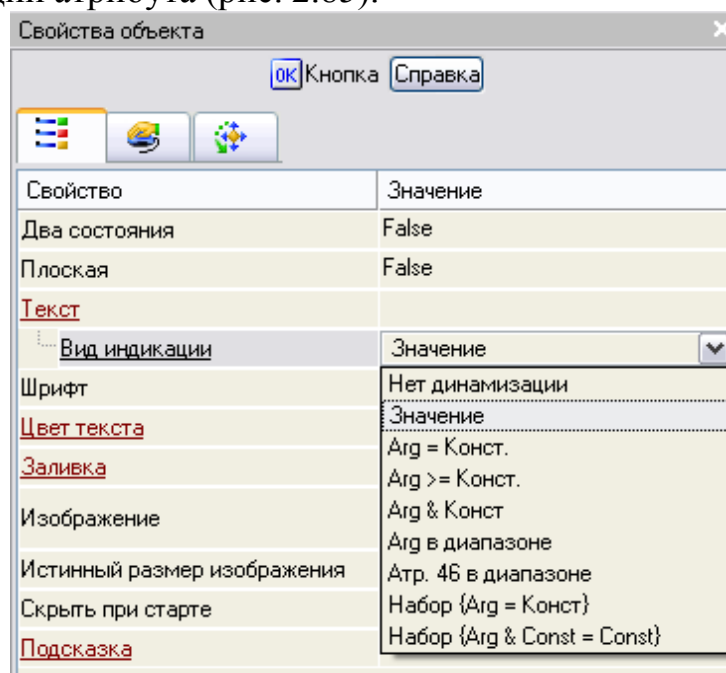


Рисунок 2.85. Список типов динамизации свойства **Текст** ГЭ *Кнопка*

С помощью ЛК выбрать тип **Значение** (рис. 2.86).



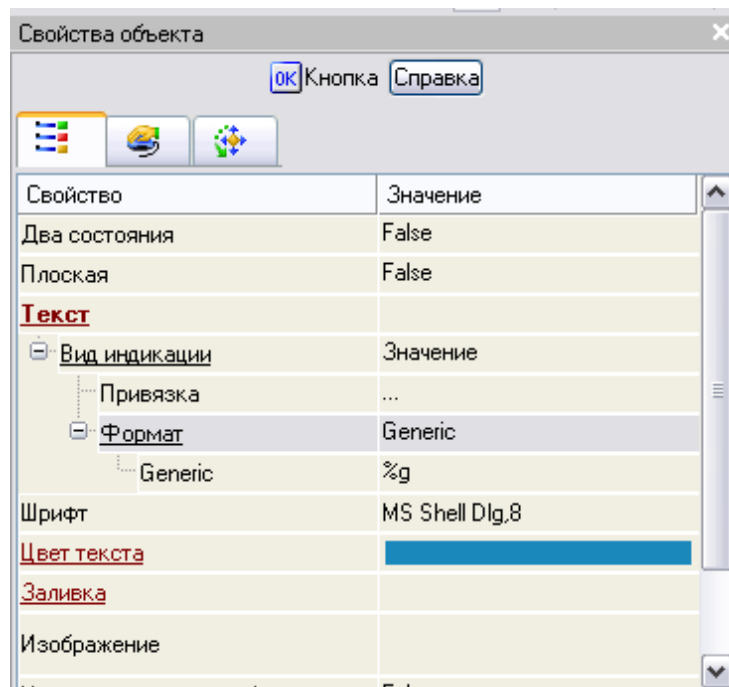


Рисунок 2.86. Установка *Вид индикации* - **Значение** свойства **Текст** ГЭ *Кнопка* Щелчком ЛК в значении поля **Привязка**, вызвать меню **Свойства привязки** (рис. 2.87).

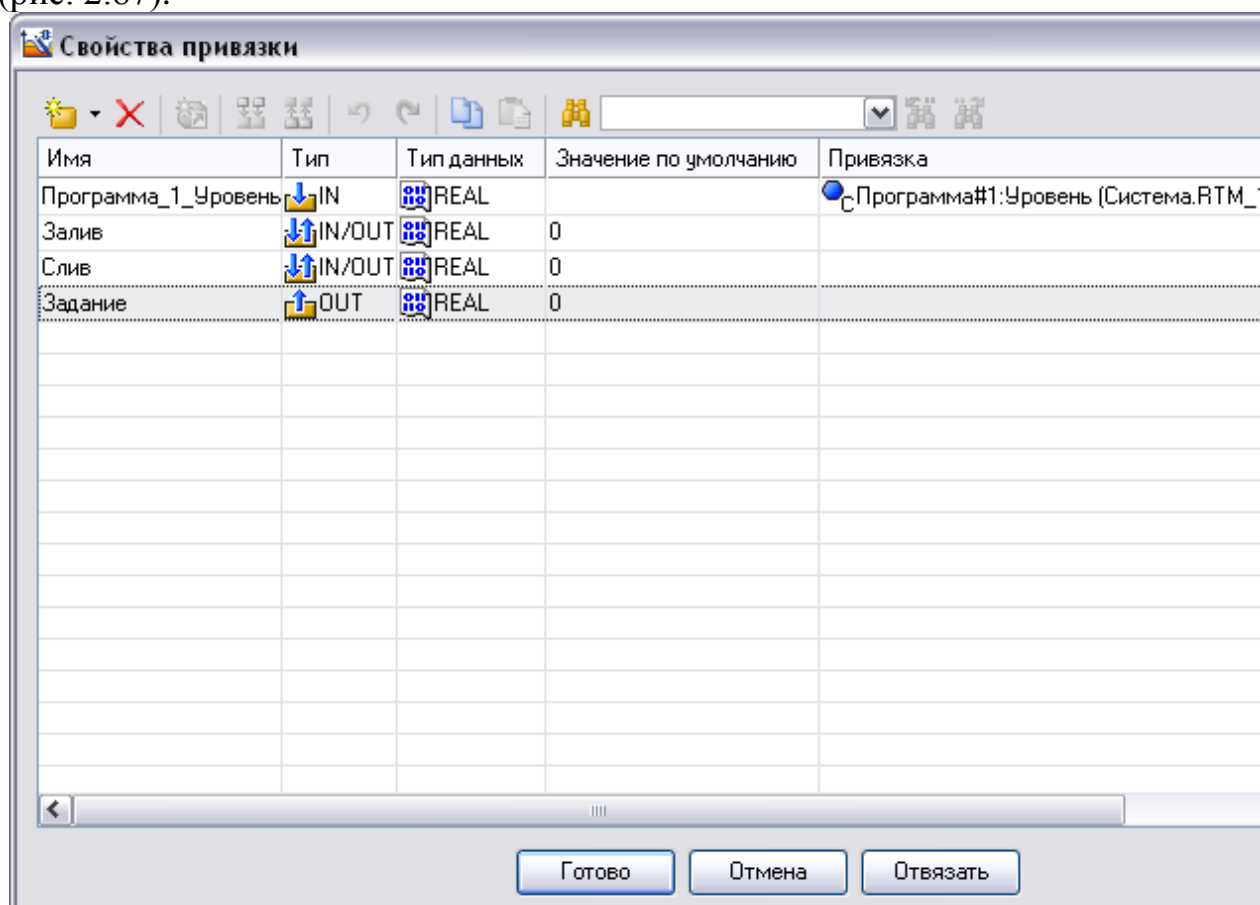


Рисунок 2.87. Свойства привязки

Выбрать ЛК аргумент **Задание** и связать ГЭ *Кнопка* с данным атрибутом нажав кнопку **Готово**.

Переключиться на бланк **Действия**  свойств ГЭ *Кнопка* (рис. 2.88).

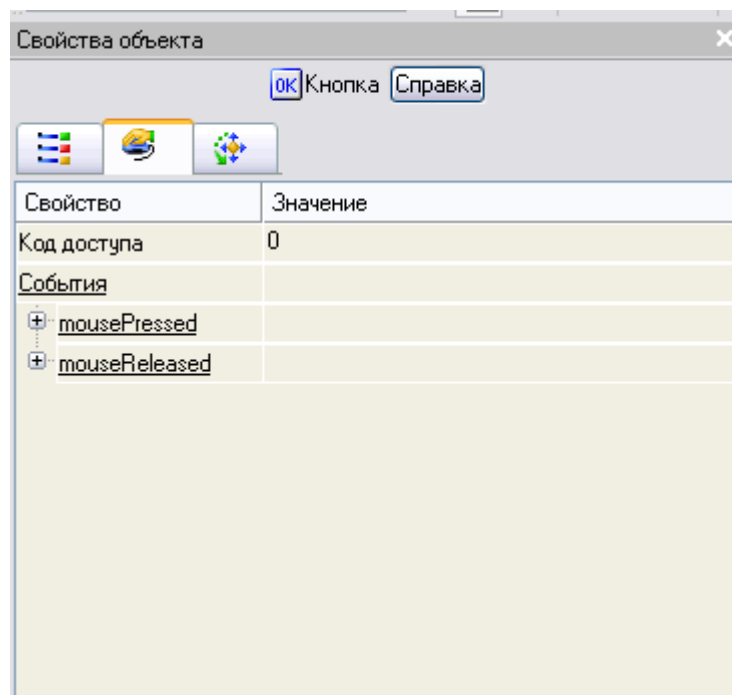


Рисунок 2.88.

ПК мыши по событию **mousePressed** вызвать контекстное меню и выбрать из списка ЛК команду **Передать значение** (рис. 2.89).

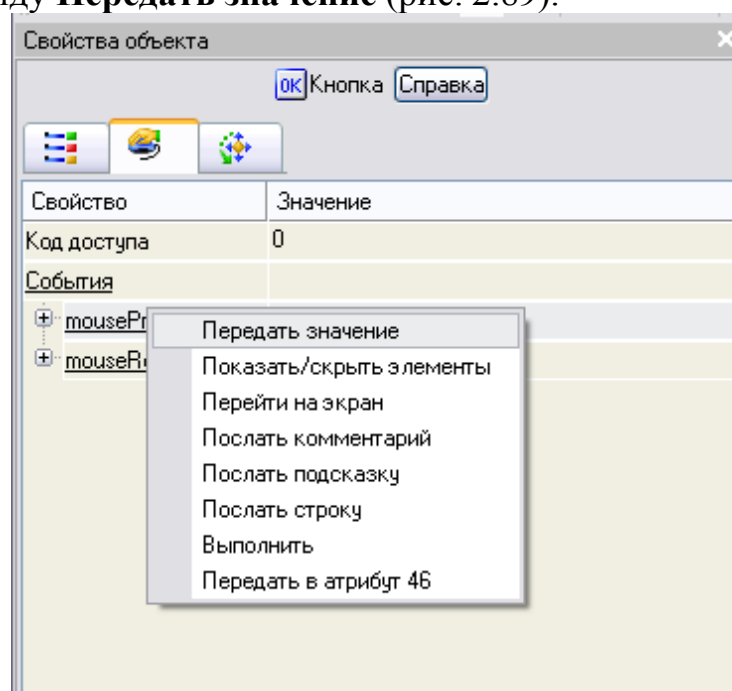


Рисунок 2.89. Создание события **mousePressed** ГЭ *Кнопка* в раскрывшемся меню настроек выбранной команды в поле **Тип передачи** выбрать из списка: **Ввести и передать** (рис. 2.90, 2.91)

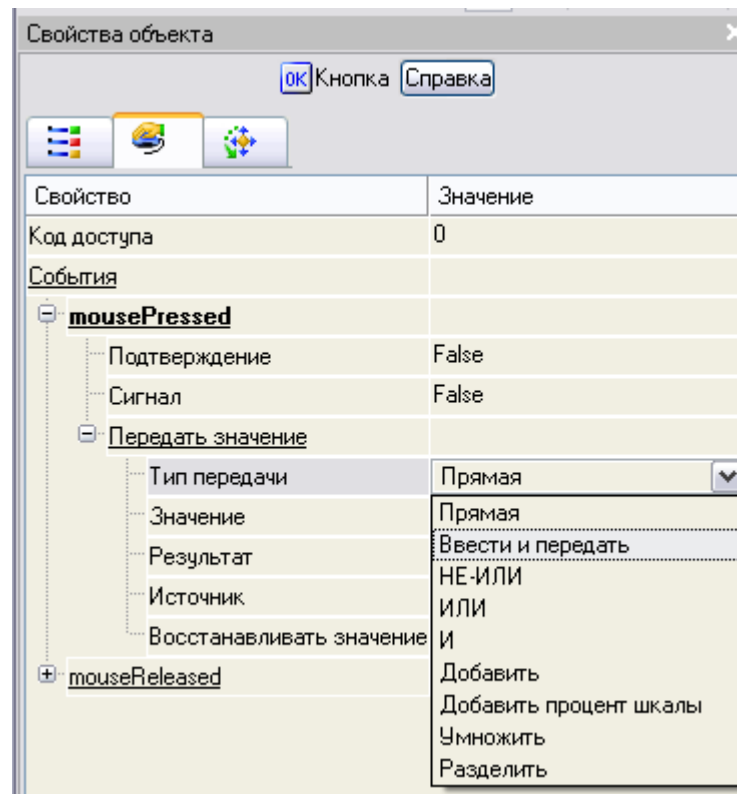


Рисунок 2.90. Список типа передачи- события **mousePressed** ГЭ *Кнопка*

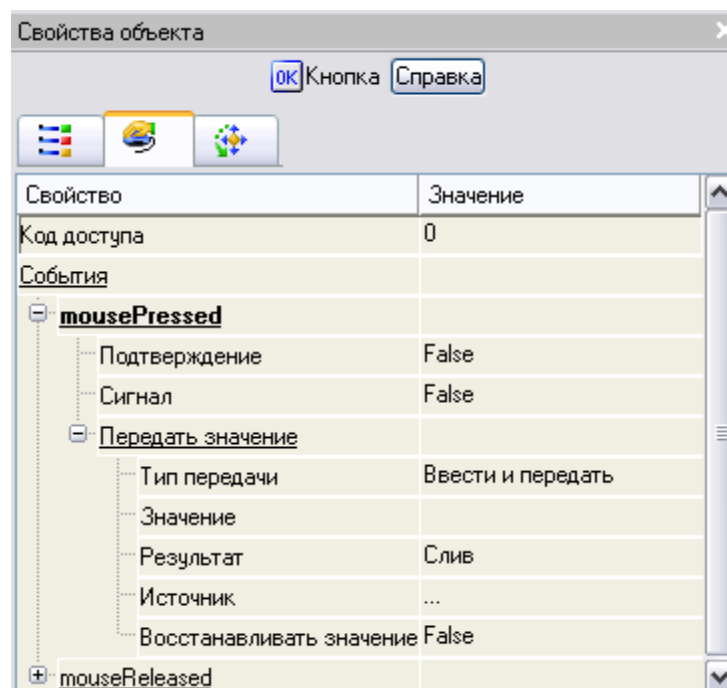


Рисунок 2.91. Установка типа передачи- **Ввести и передать** - события **mousePressed** ГЭ *Кнопка*

щелчком ЛК в поле **Результат** вызвать табличный редактор аргументов (рис. 2.92).

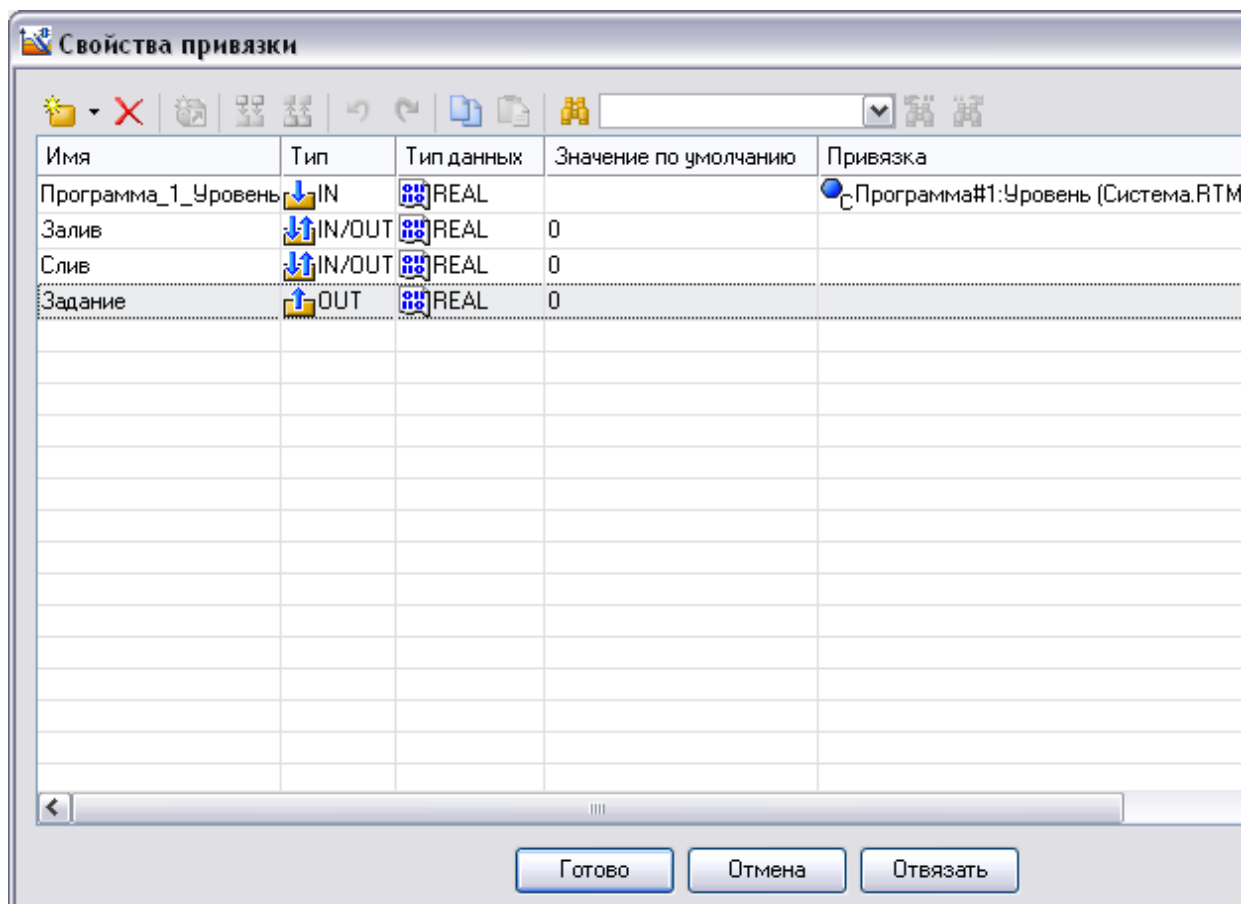


Рисунок 2.91. Свойства привязки

Выбрать аргумент **Задание** и привязать его к ГЭ кнопкой **Готово**

На панели инструментов графического редактора щелчком ЛК по кнопке **АВС** вызвать ГЭ **Текст** (рис. 2.92).



Рисунок 2.92. Панель инструментов графического редактора

Затем, в поле графического редактора, ЛК указать левый верхний угол ГЭ, движением мыши растянуть до необходимого размера и зафиксировать правый нижний угол ЛК (рис. 2.93).

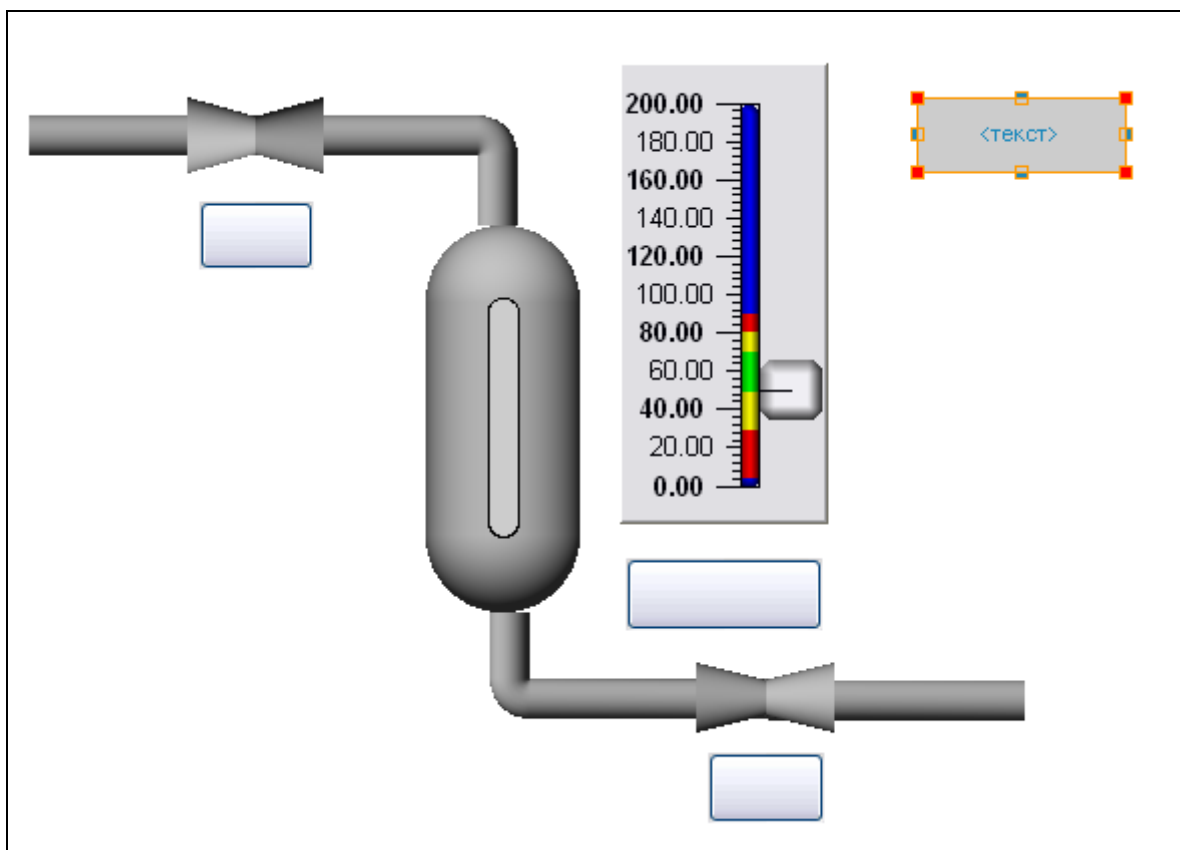



Рисунок 2.93. Создание ГЭ *Текст*

Перейти в режим редактирования атрибутов ГЭ *Текст*, выделив ЛК иконку  на панели инструментов и двойным щелчком ЛК по ГЭ открыть его свойства (рис. 2.94).

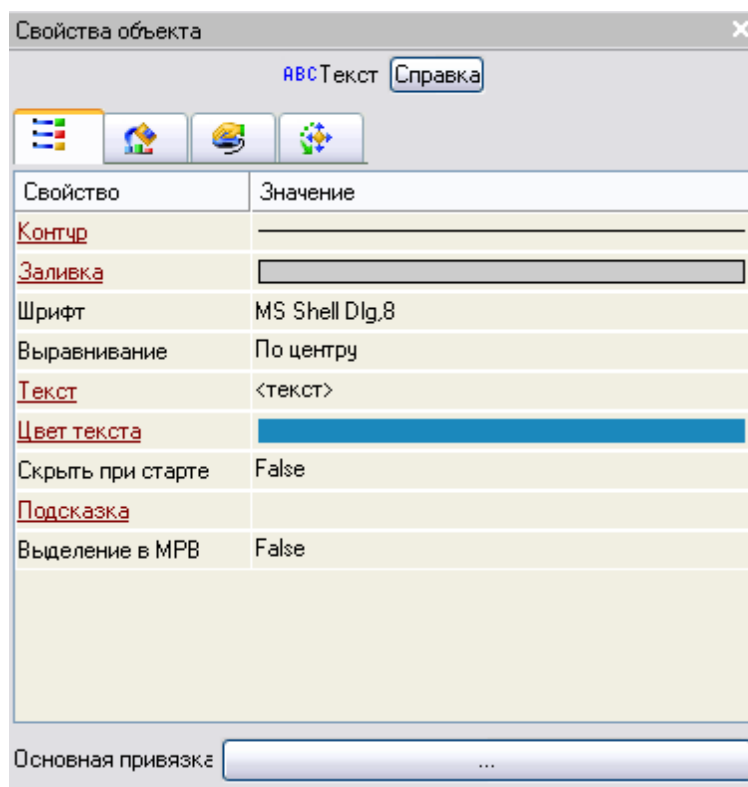


Рисунок 2.94. Свойства ГЭ *Текст*

Щелкнуть по значению свойства **Текст**, набрать *Зона* и подтвердить ввод нажатием **Enter** (рис. 2.95).

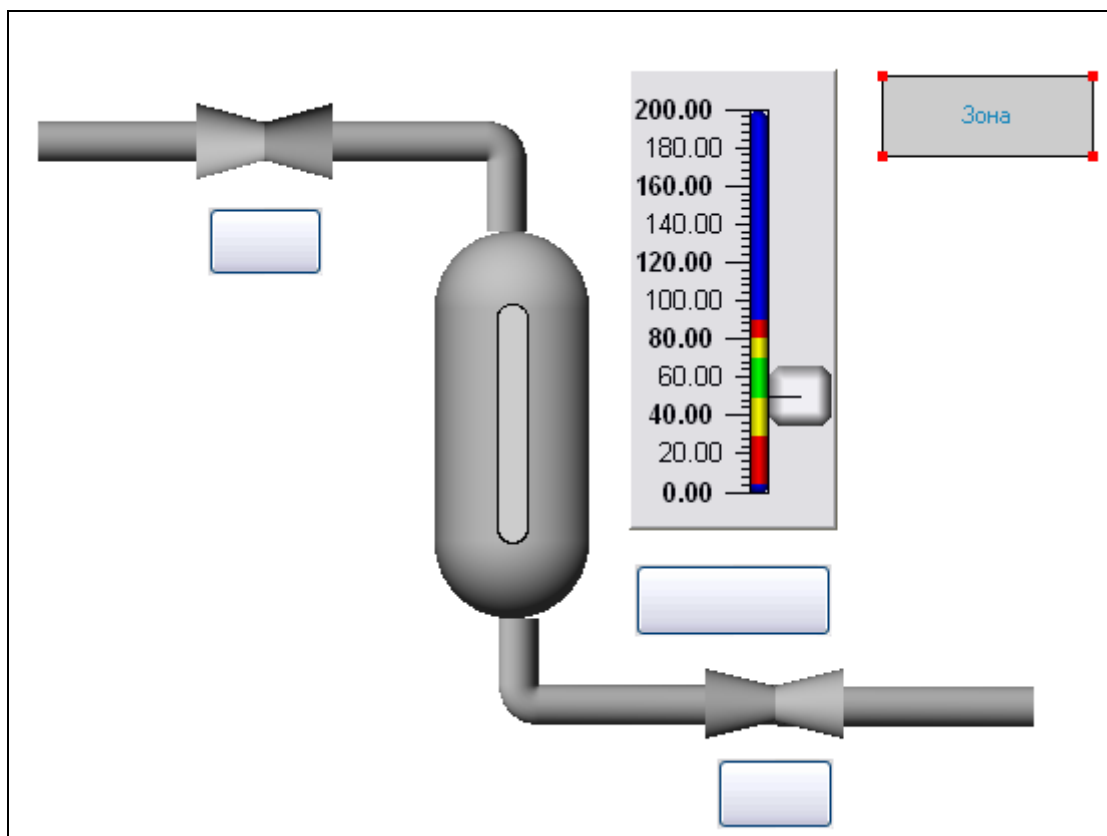



Рисунок 2.95. Изменение ГЭ *Текст*

Для задания зоны нечувствительности регулятора на инструментальной панели графического редактора выбрать ЛК иконку ГЭ *Кнопка* . С помощью мыши разместить его напротив ГЭ *Текст Зона* (рис. 2.96).

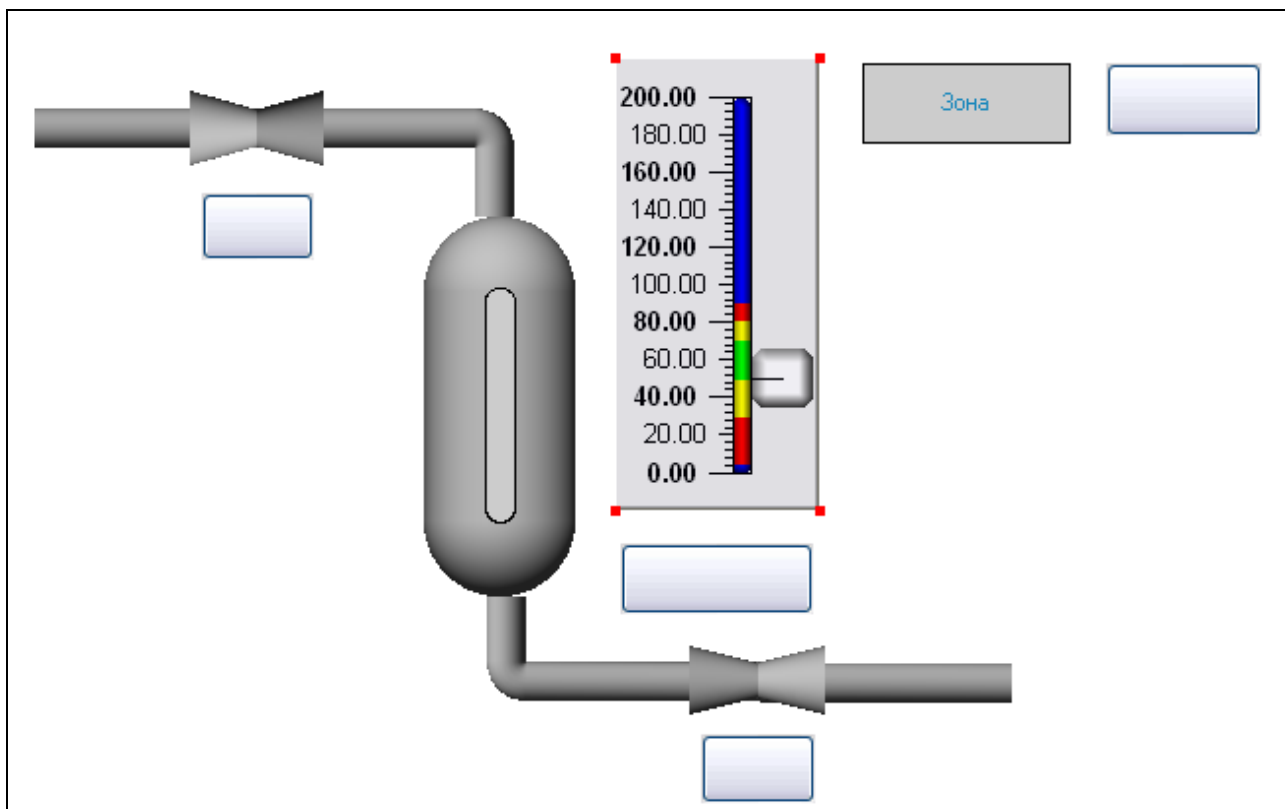



Рисунок 2.96. Создание ГЭ *Кнопка*

Перейти в режим редактирования, выделив ЛК иконку  на панели инструментов.

Щелчком ЛК по ГЭ **Кнопка** вызвать окно его свойств (рис. 2.97).

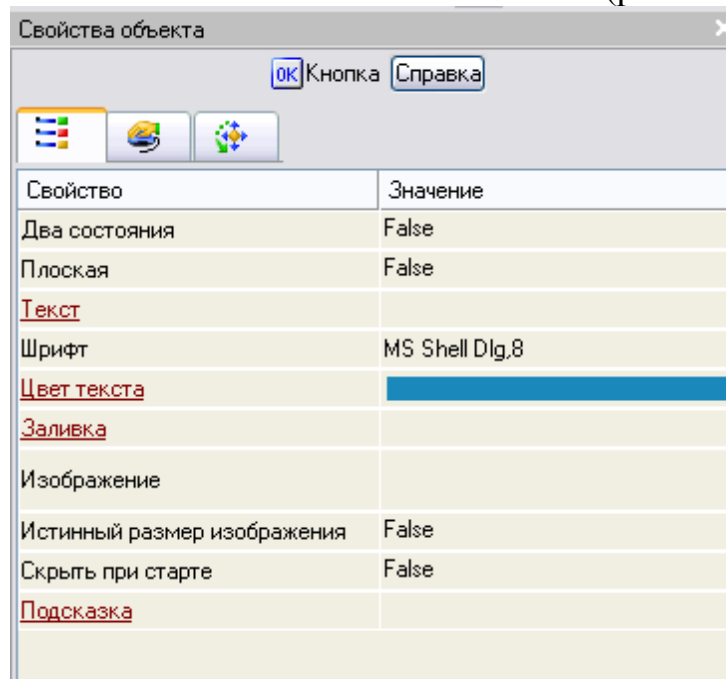


Рисунок 2.97. Свойства ГЭ *Кнопка*

Двойным щелчком ЛК на строке **Текст** вызвать меню **Вид индикации** (рис. 2.98).

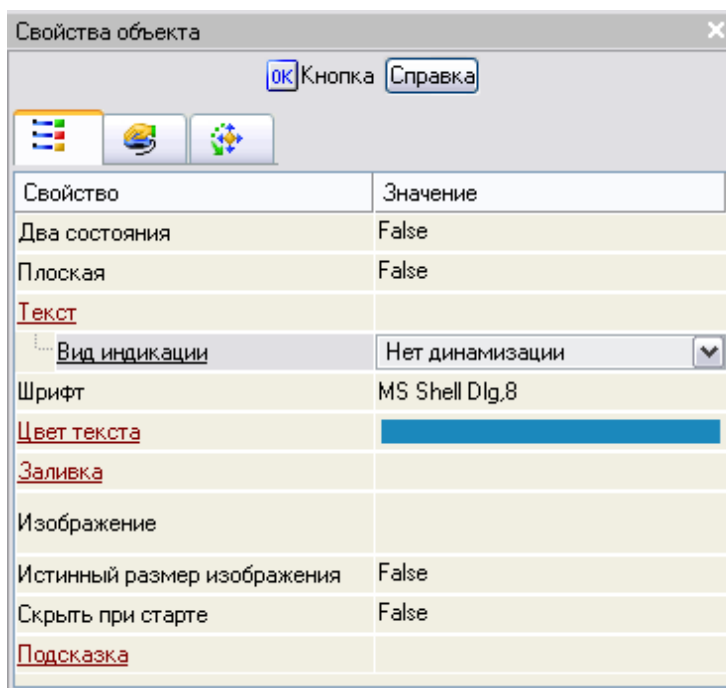


Рисунок 2.98. Вид индикации ГЭ *Кнопка*

Щелчком ЛК по значению меню **Вид индикации** вызвать список доступных типов динамизации атрибута (рис. 2.99).

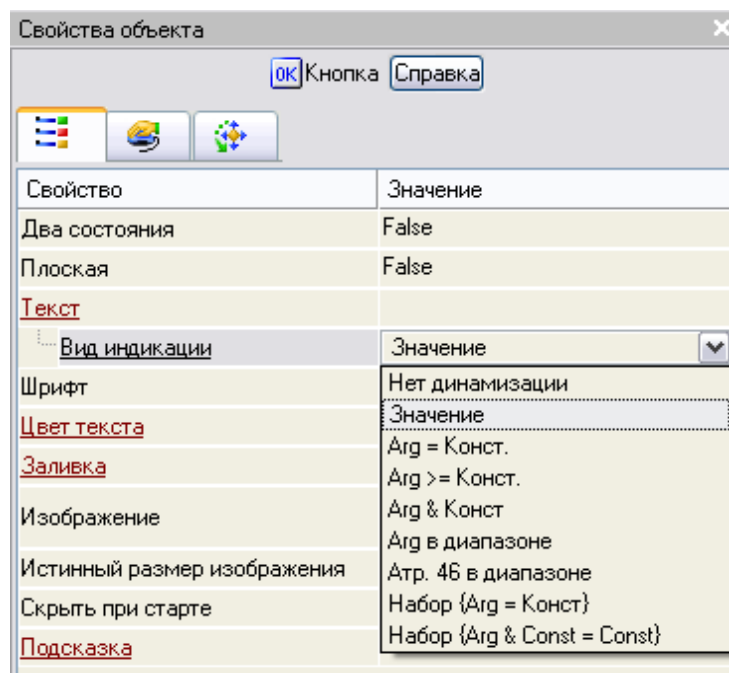


Рисунок 2.99. Список типов динамизации ГЭ *Кнопка*  
С помощью ЛК выбрать тип **Значение** (рис. 2.100).

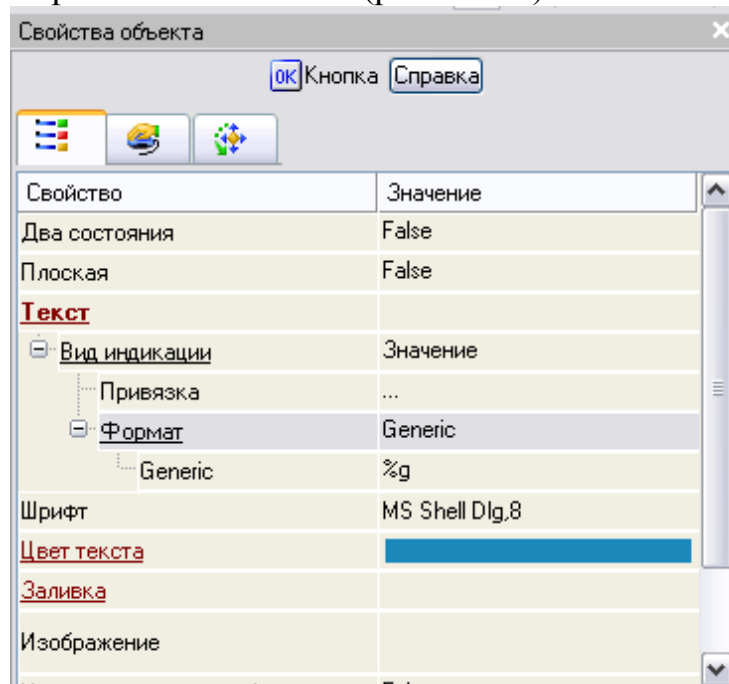
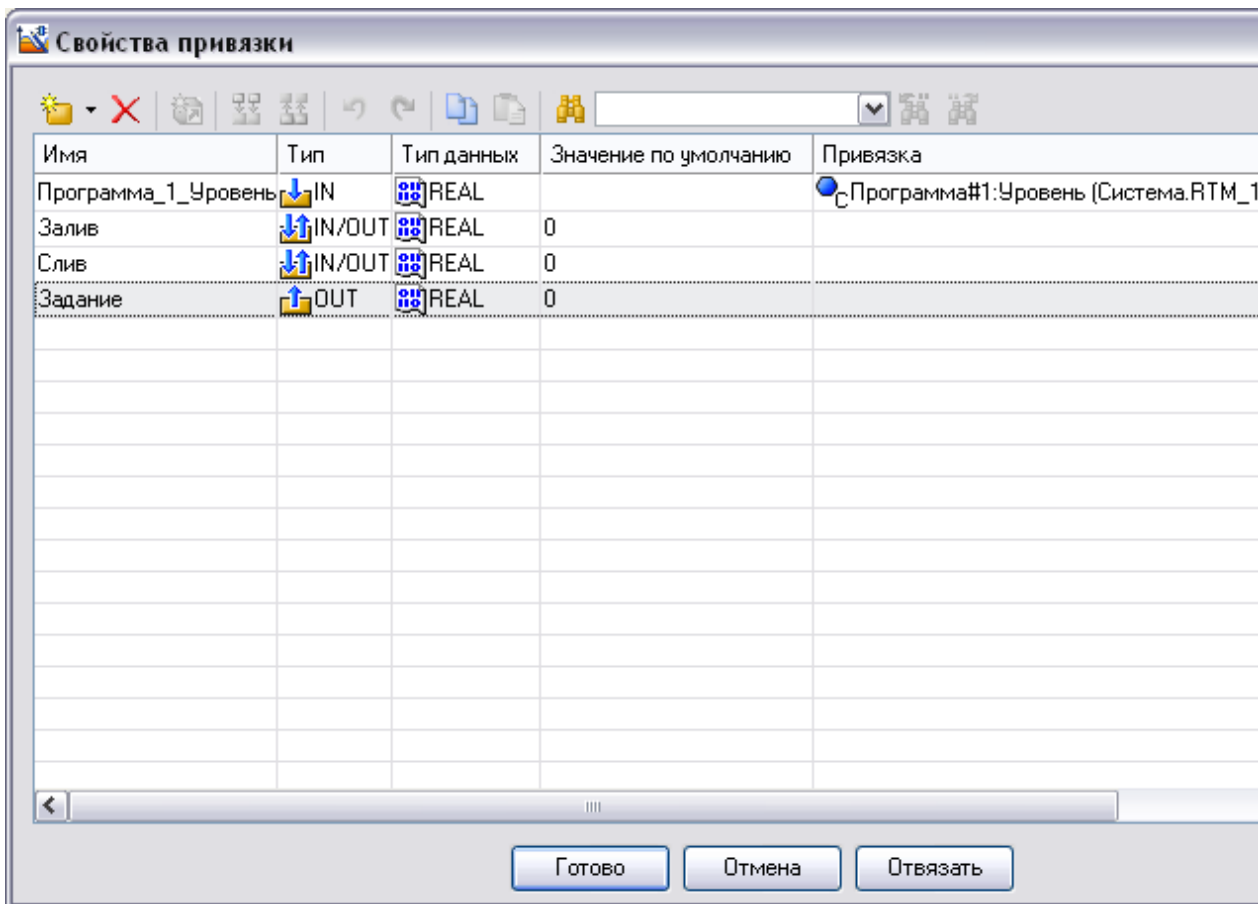



Рисунок 2.100. Установка *Вид индикации* - **Значение** свойства **Текст** ГЭ *Кнопка*  
Щелчком ЛК в значении поля **Привязка**, вызвать меню **Свойства привязки**  
(рис. 2.101)





### Рисунок 2.101. Свойства привязки

Выбрать ЛК аргумент **Задание** и связать ГЭ *Кнопка* с данным атрибутом нажав кнопку **Готово**.

Щелчком ЛК по кнопке  создать новый аргумент (рис. 2.102).

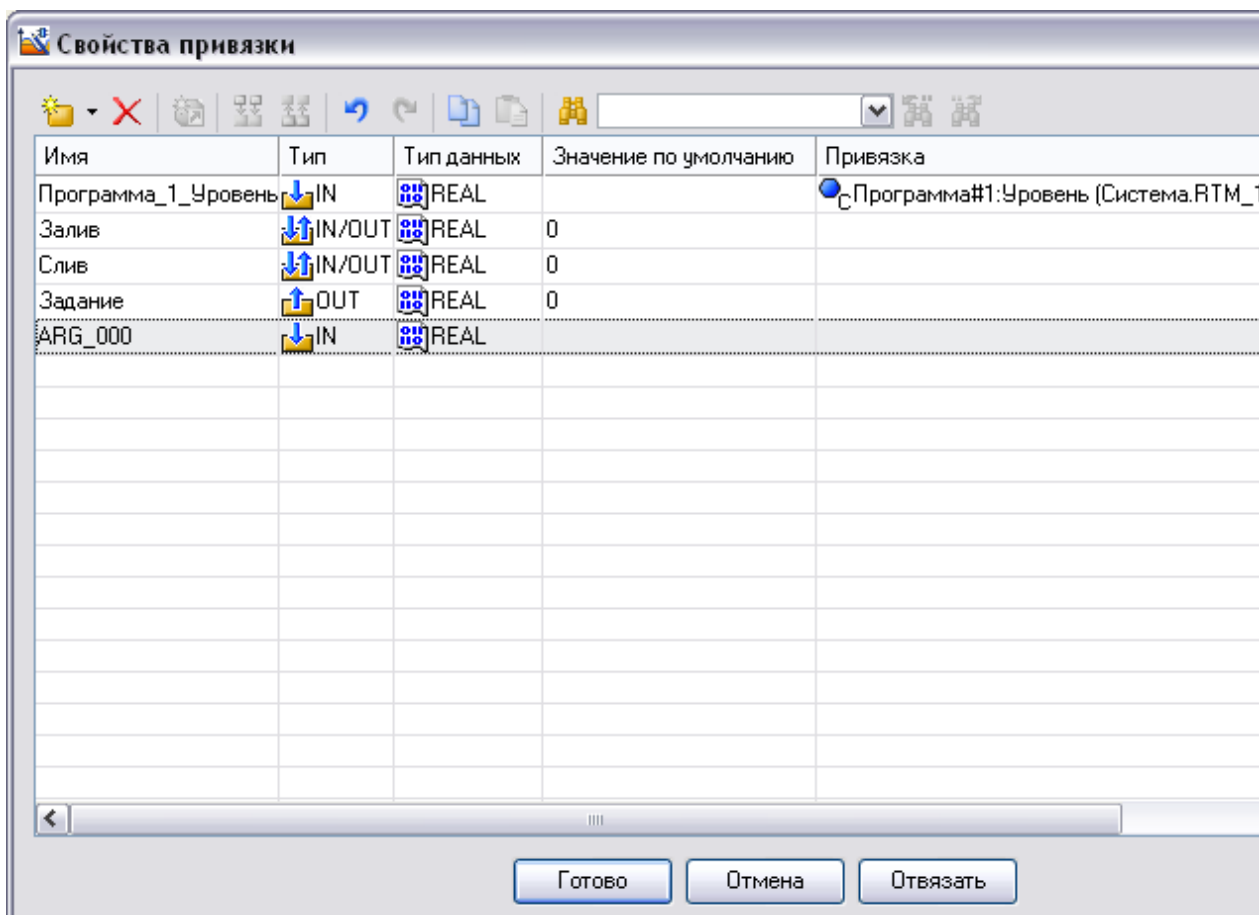


Рисунок 2.102. Свойства привязки – создание аргумента

Двойным щелчком ЛК зайти в ячейку *Имя* аргумента и изменить его изменить на **Зона** (введя имя с клавиатуры, завершив ввод нажатием клавиши **Enter**)  
Двойным щелчком ЛК по ячейке *Тип* аргумента вызвать выпадающий список, в котором выбрать тип **OUT** (рис. 2.103).

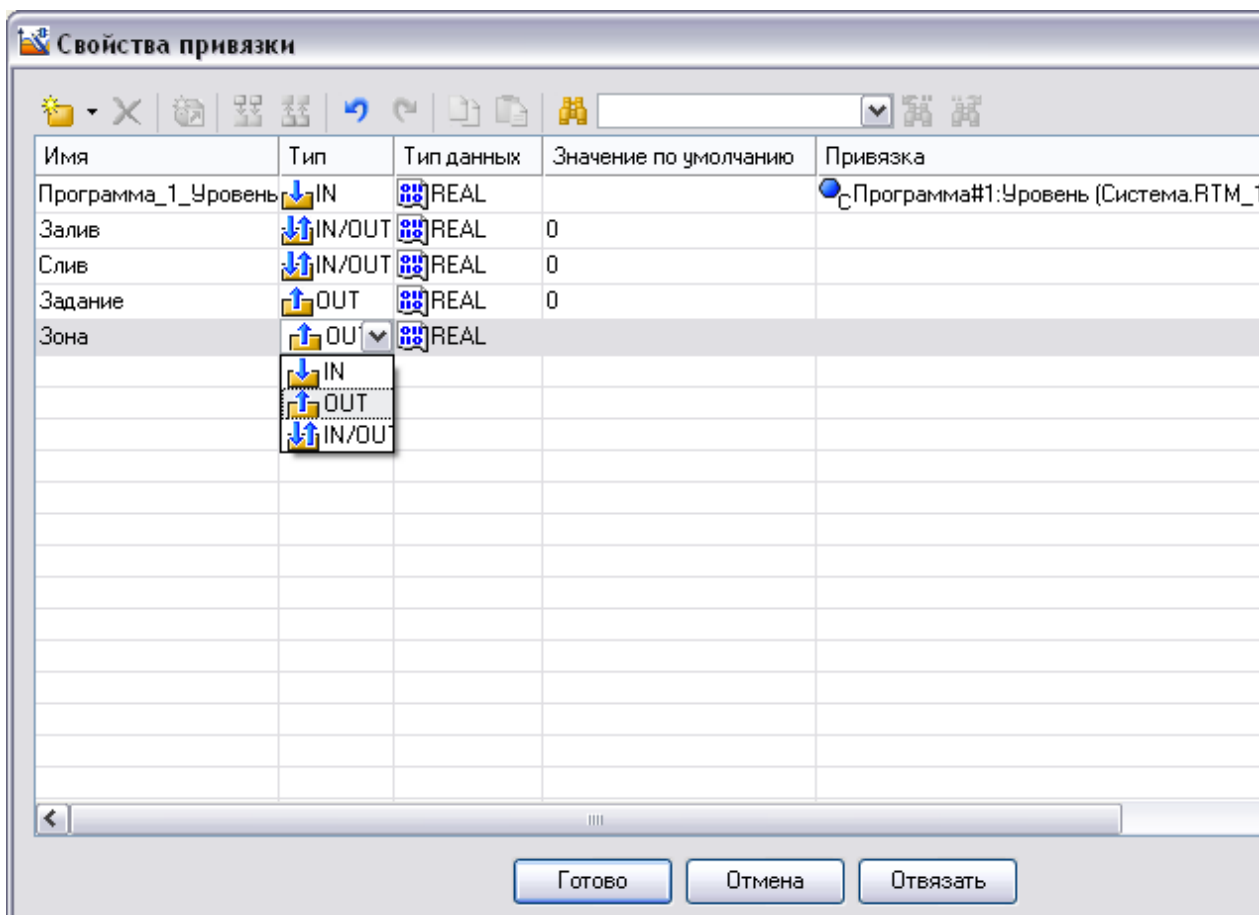


Рисунок 2.103. Свойства привязки – установка типа аргумента  
Связать ГЭ *Кнопка* с данным атрибутом нажав кнопку **Готово**.

Переключиться на бланк **Действия**  свойств ГЭ *Кнопка* (рис. 2.104)

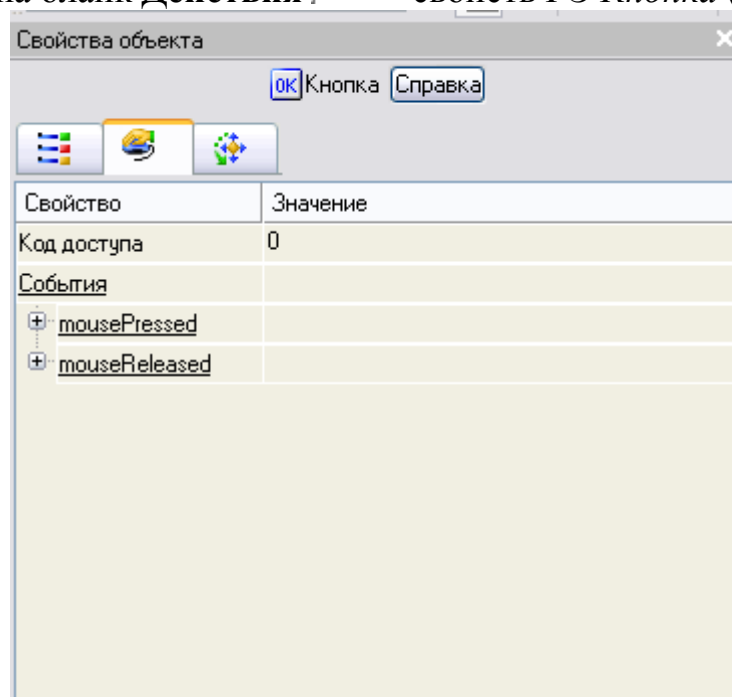


Рисунок 2.104. Бланк **Действия** свойств ГЭ *Кнопка*  
ПК мыши по событию **mousePressed** вызвать контекстное меню и выбрать из списка ЛК команду **Передать значение** (рис. 2.105)

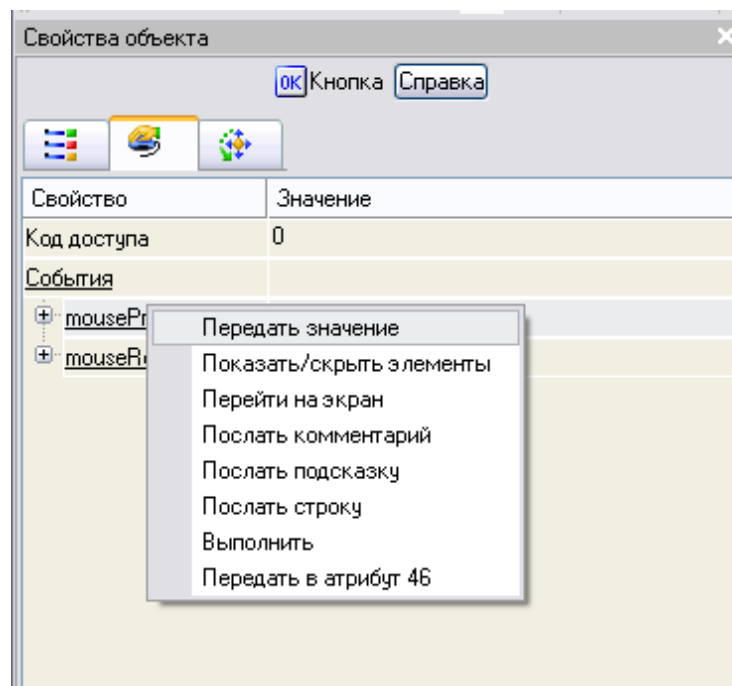


Рисунок 2.105. Создание события **mousePressed** ГЭ *Кнопка* в раскрывшемся меню настроек выбранной команды в поле **Тип передачи** выбрать из списка **Ввести и передать** (рис. 2.106, 2.107)

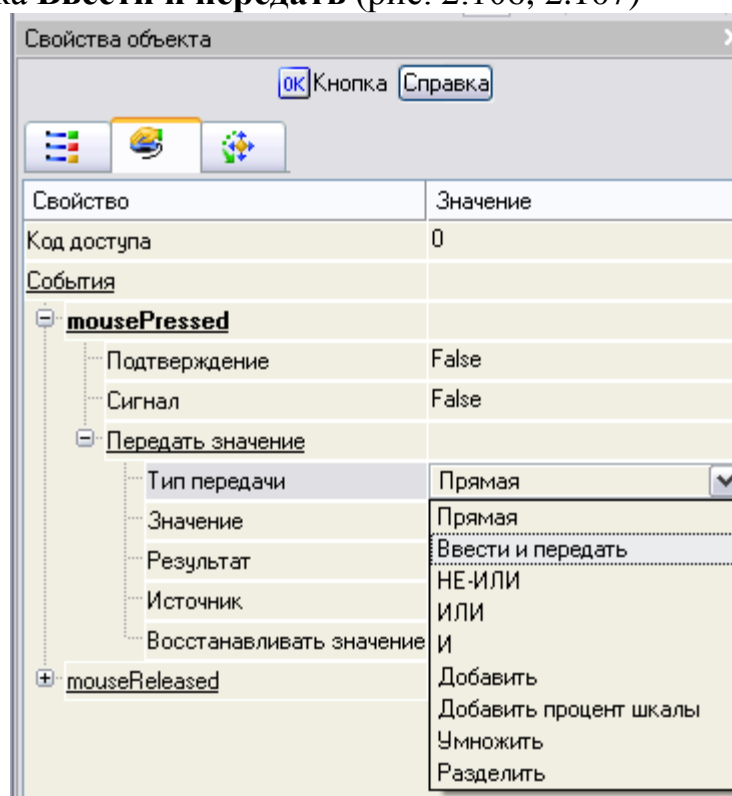


Рисунок 2.106. Список **Типов передачи** события **mousePressed** ГЭ *Кнопка*

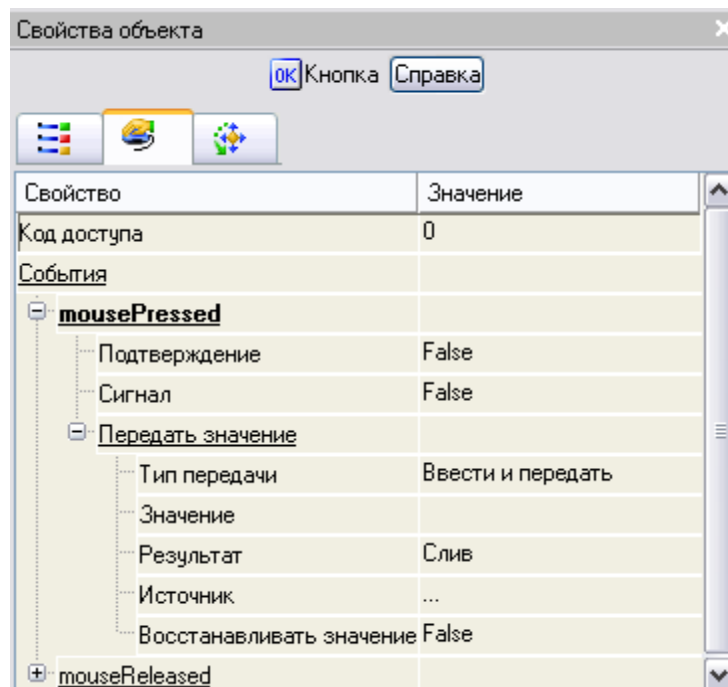


Рисунок 2.107. Установка типа передачи- **Ввести и передать** - события **mousePressed** ГЭ *Кнопка*

щелчком ЛК в поле **Результат** вызвать табличный редактор аргументов (рис. 2.108)

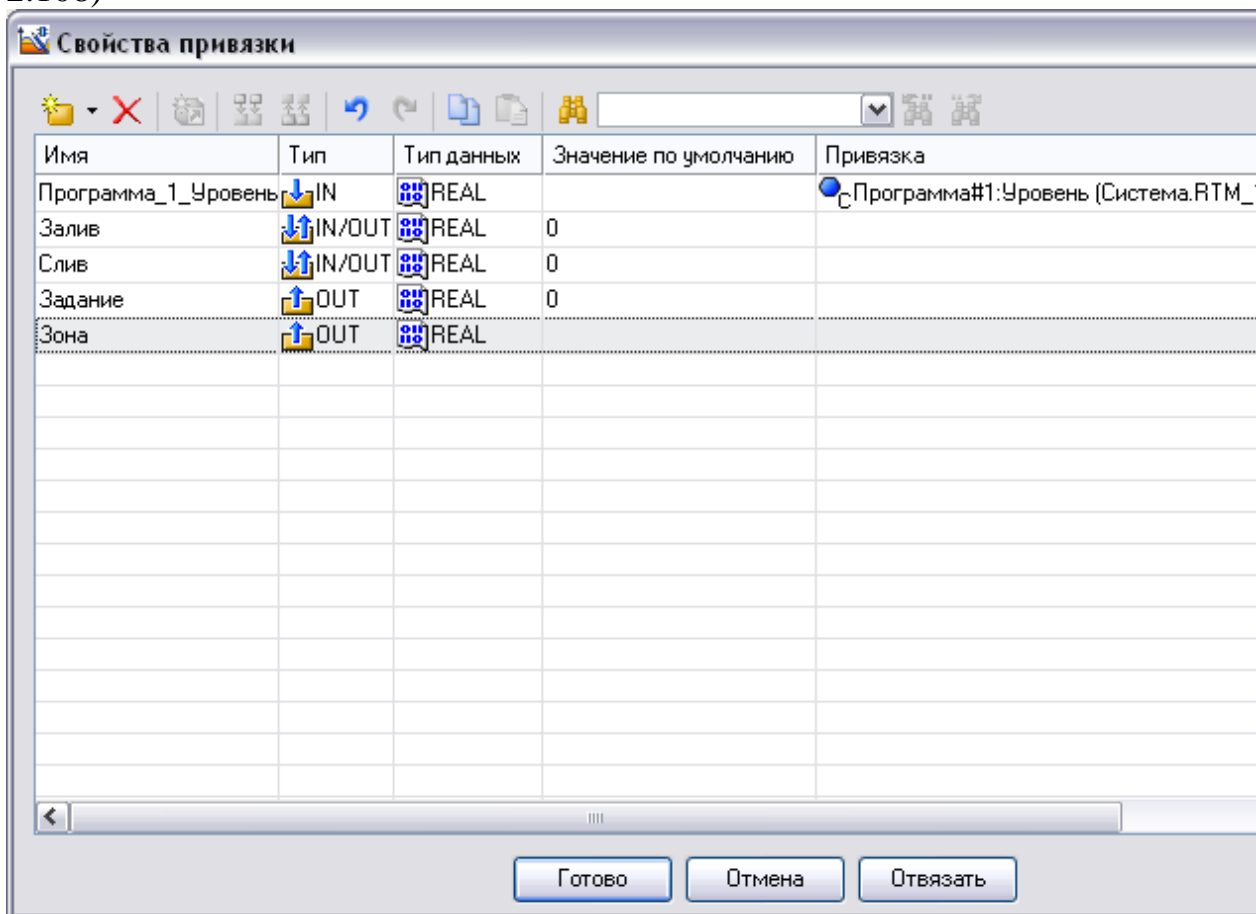


Рисунок 2.108. Свойства привязки

Выбрать аргумент **Зона** и привязать его к ГЭ кнопкой **Готово**

На панели инструментов графического редактора щелчком ЛК по кнопке



вызвать ГЭ *Текст* (рис. 2.109).



Рисунок 2.109. Панель инструментов графического редактора

Затем, в поле графического редактора, ЛК указать левый верхний угол ГЭ, движением мыши растянуть до необходимого размера и зафиксировать правый нижний угол ЛК (рис. 2.110).

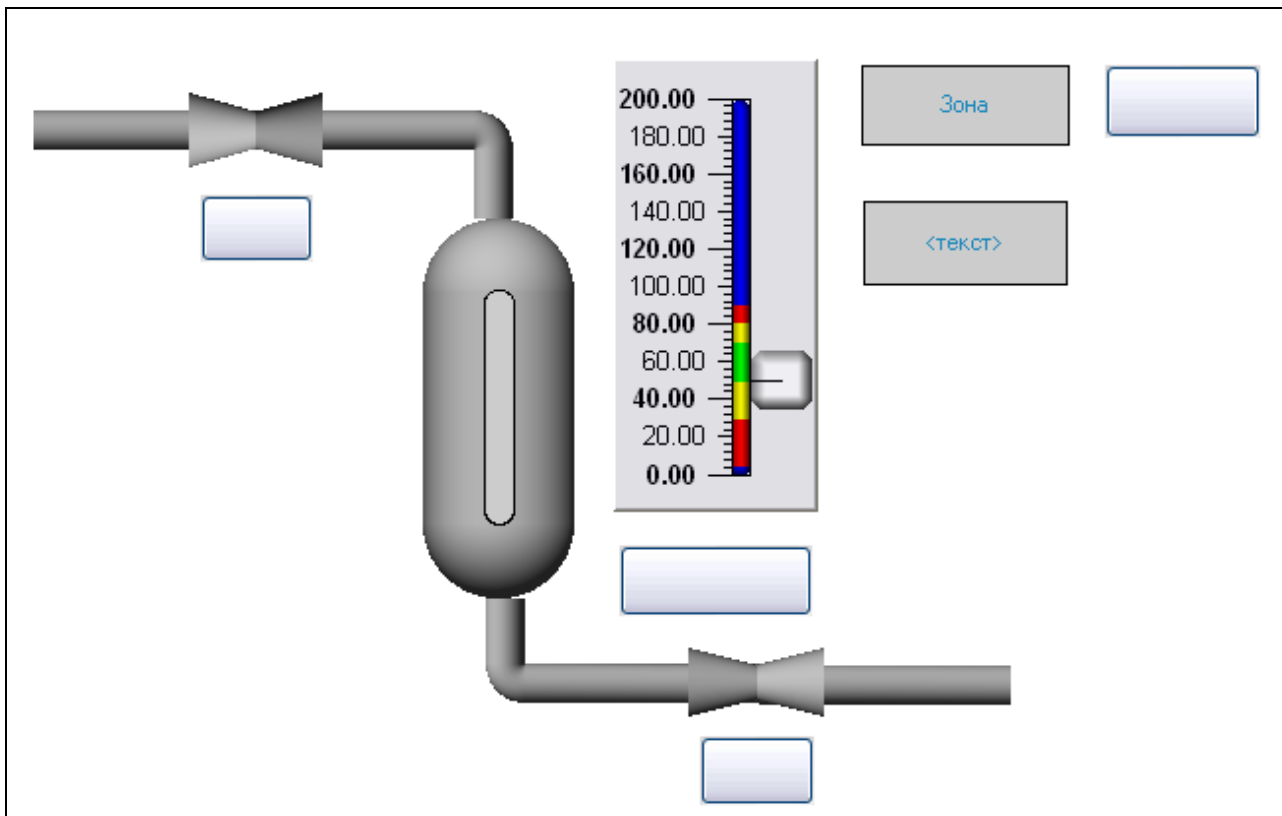



Рисунок 2.110. Создание ГЭ *Текст*

Перейти в режим редактирования атрибутов ГЭ *Текст*, выделив ЛК иконку  на панели инструментов и двойным щелчком ЛК по ГЭ открыть его свойства (рис. 2.111).

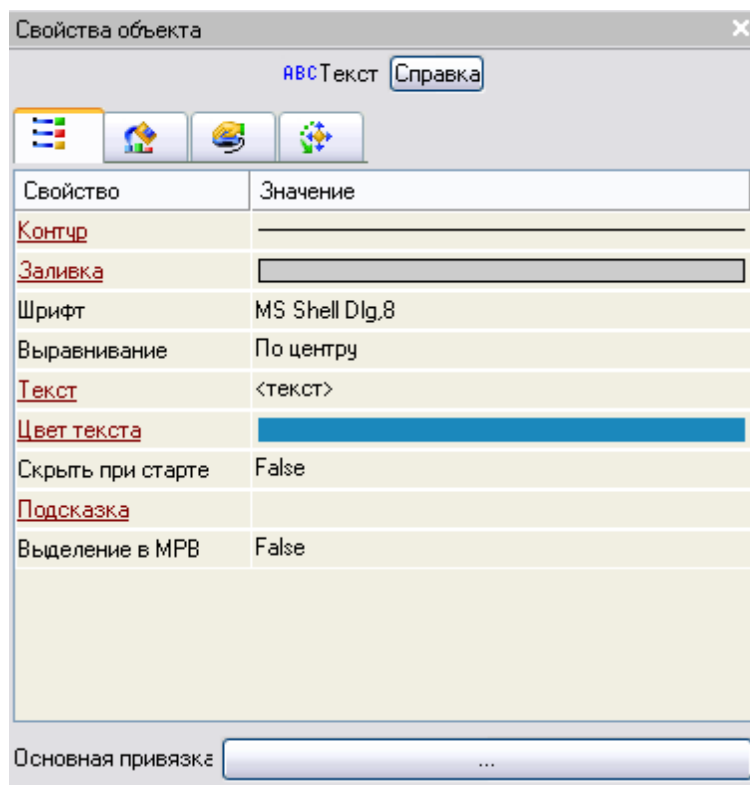


Рисунок 2.111. Свойства ГЭ *Текст*

Щелкнуть по значению свойства **Текст**, набрать *Уровень* и подтвердить ввод нажатием **Enter** (рис. 2.112).

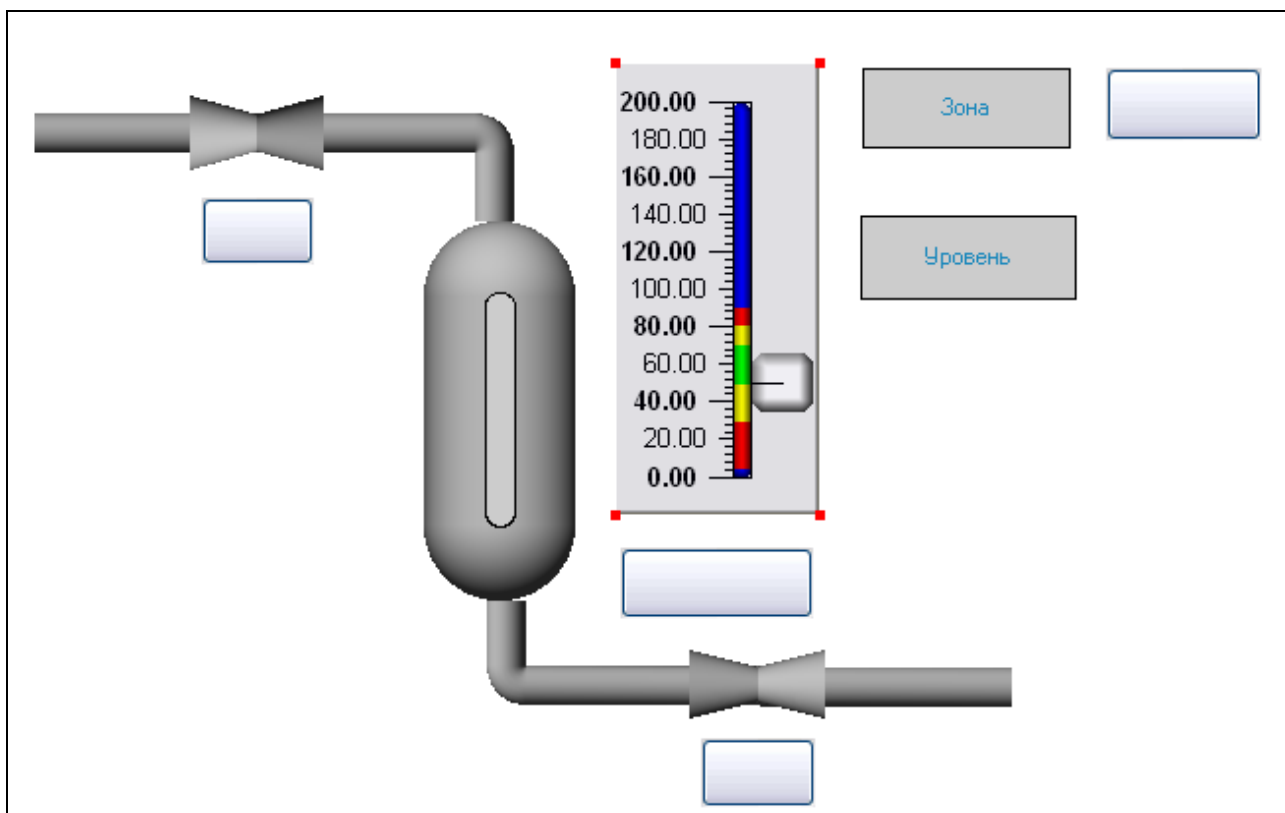


Рисунок 2.112. Изменение ГЭ *Текст*

На панели инструментов графического редактора щелчком ЛК по кнопке ABC вызвать ГЭ *Текст* (рис. 2.113)



Рисунок 2.113. Панель инструментов графического редактора  
, затем, в поле графического редактора, ЛК указать левый верхний угол ГЭ, движением мыши растянуть до необходимого размера и зафиксировать правый нижний угол ЛК.

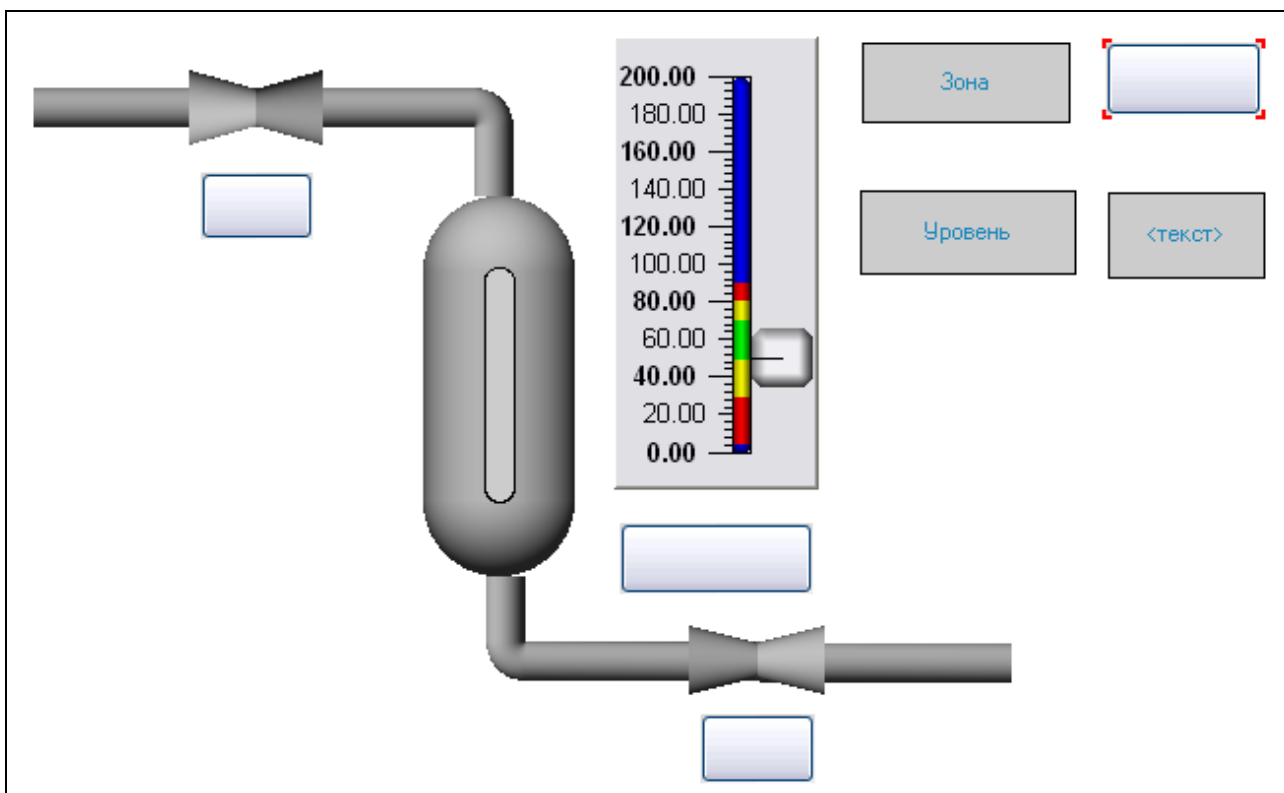



Рисунок 2.114. Создание ГЭ *Текст*

Перейти в режим редактирования атрибутов ГЭ *Текст*, выделив ЛК иконку  на панели инструментов и двойным щелчком ЛК по ГЭ открыть его свойства (рис. 2.115).

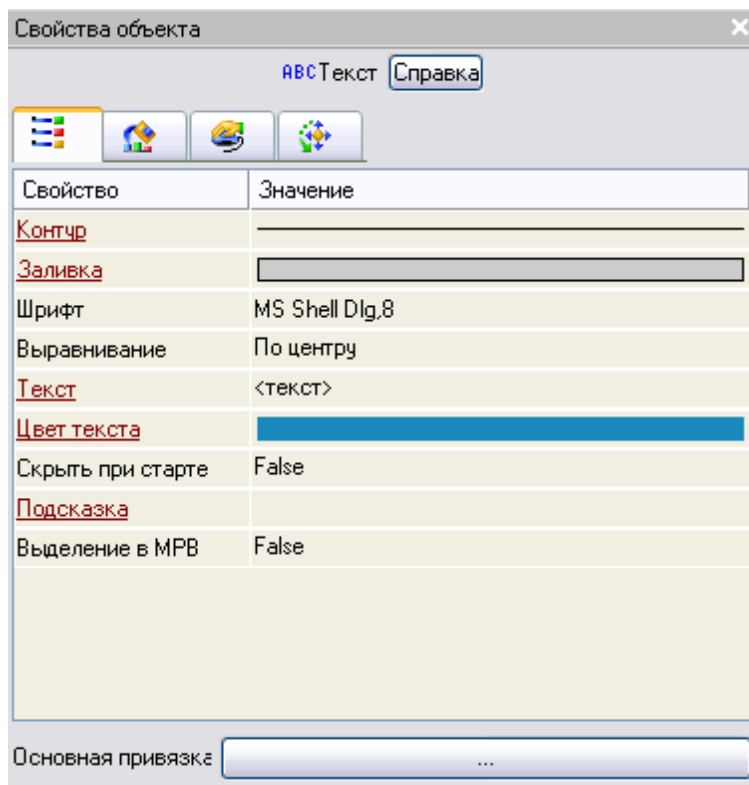




Рисунок 2.115. Свойства ГЭ *Текст*

Двойным щелчком ЛК на свойстве *Текст* вызовем меню **Вид индикации** (рис. 2.116).

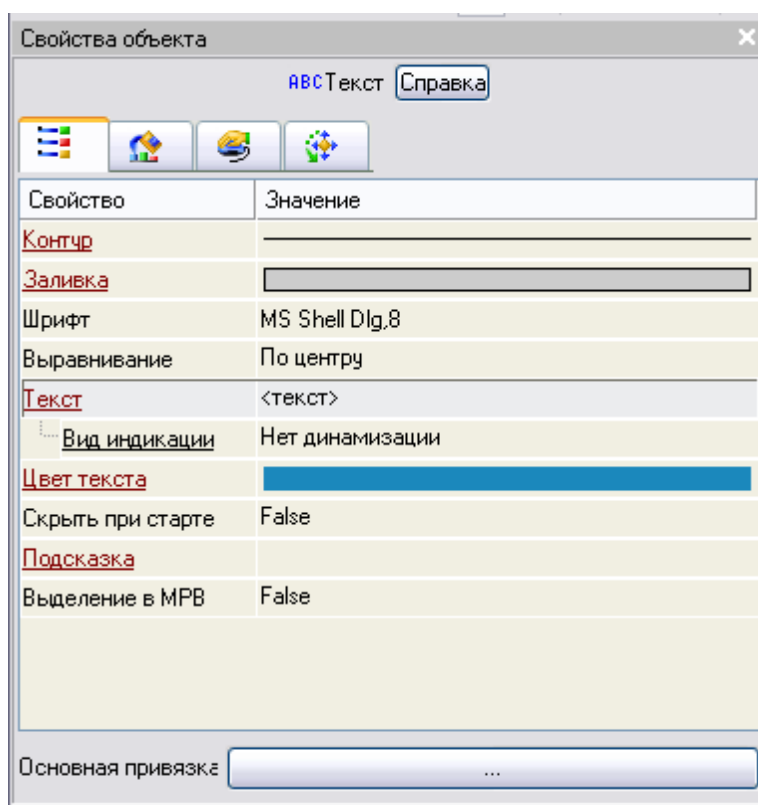


Рисунок 2.115. Свойство **Вид индикации** ГЭ *Текст*

Щелчком ЛК по значению **Вид индикации** (в правом поле строки) вызвать список доступных типов динамизации атрибута (рис. 2.116)

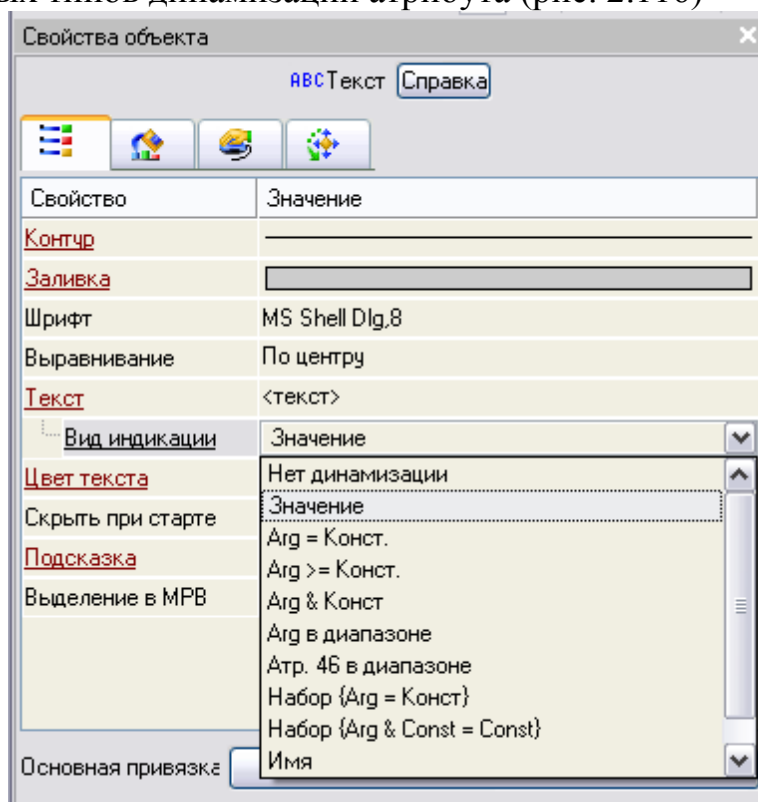


Рисунок 2.116. Список типов динамизации свойства **Вид индикации** ГЭ *Текст* из всех предлагаемых типов выбрать ЛК **Значение** (рис. 2.117).

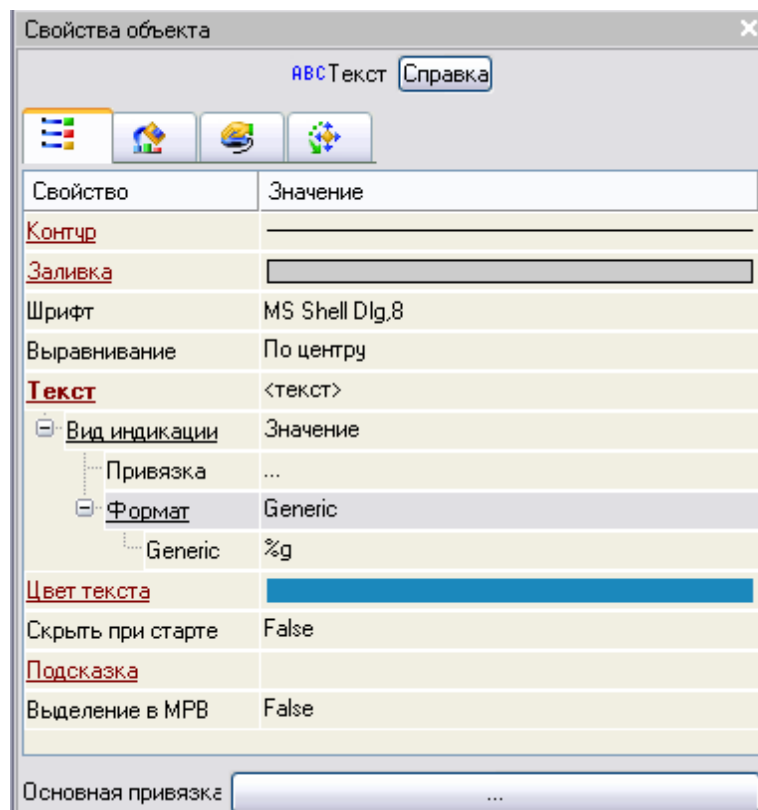


Рисунок 2.117. Установка Вид индикации - Значение свойства Текст ГЭ Текст  
В меню настроек динамизации вызвать **Свойства привязки** щелчком ЛК в правом поле строки **Привязка** (рис. 2.118)

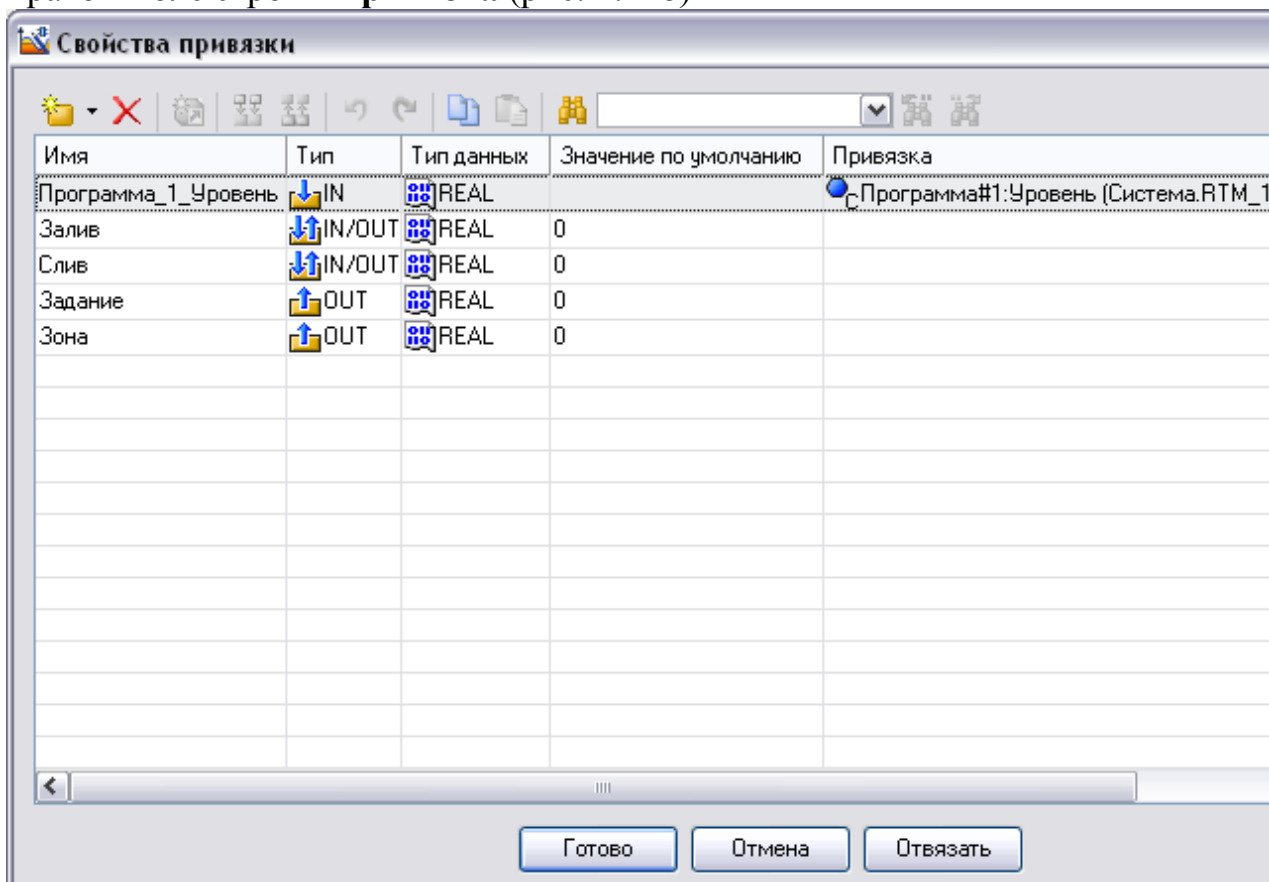

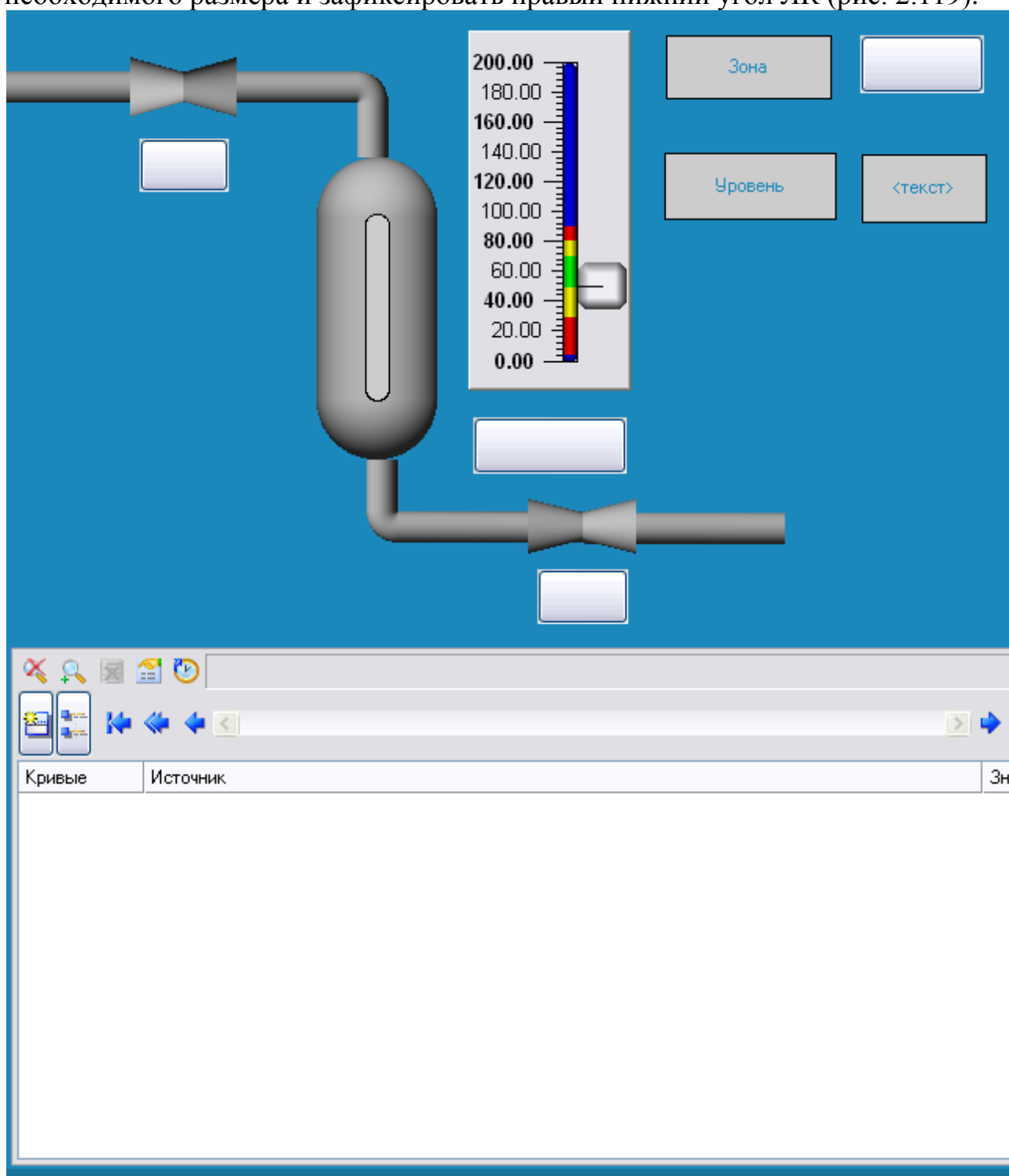


Рисунок 2.118. Свойства привязки  
ЛК выбрать аргумент **Программа\_1\_Уровень** и связать его с ГЭ Текст нажав кнопку **Готово**.

Щелчком ЛК по кнопке  выделить ГЭ *Тренды*, затем, в поле графического редактора, ЛК указать левый верхний угол ГЭ, движением мыши растянуть до необходимого размера и зафиксировать правый нижний угол ЛК (рис. 2.119).



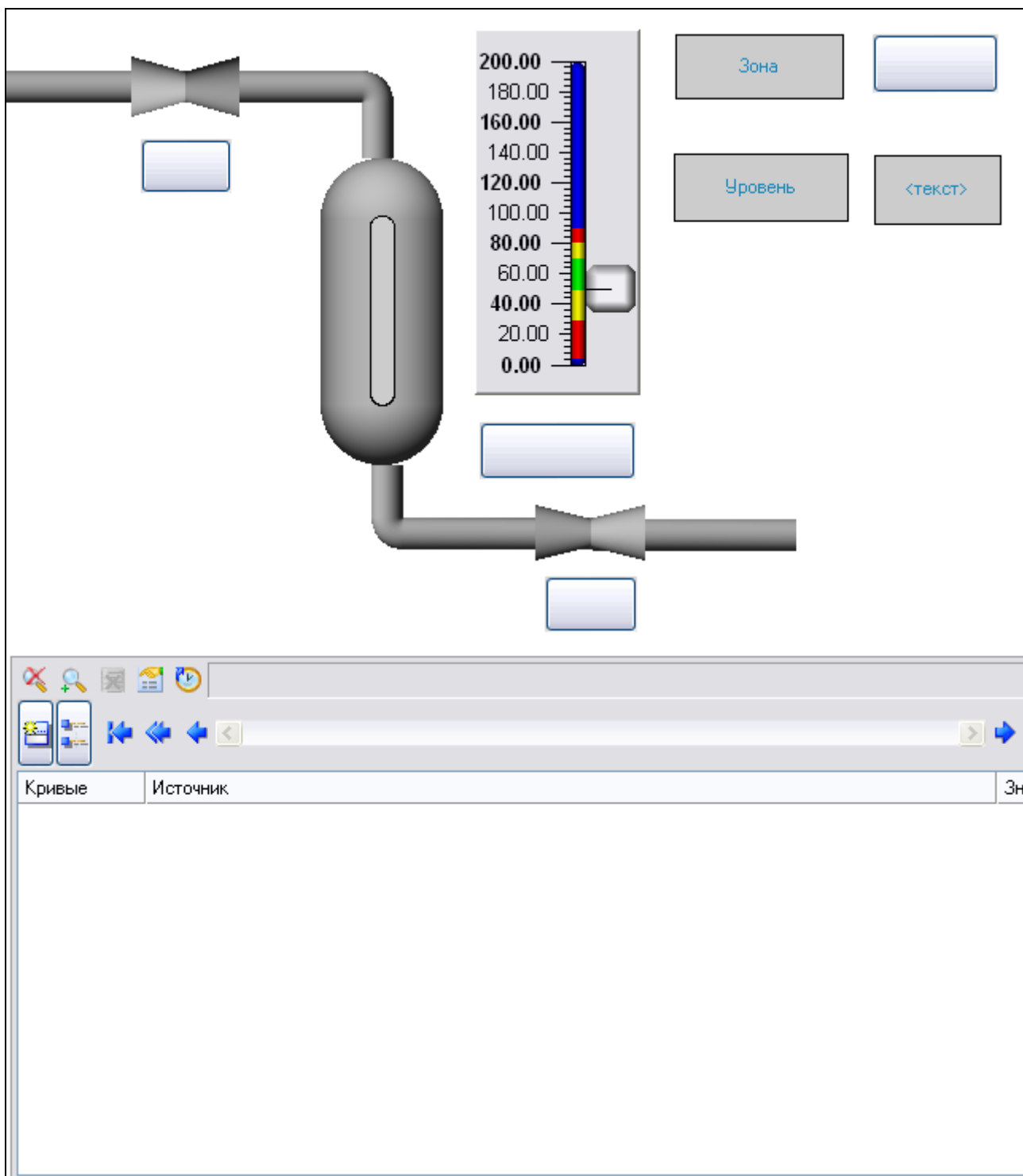



Рисунок 2.119. Создание ГЭ *Тренд*

Перейти в режим редактирования атрибутов ГЭ **Тренды**, выделив ЛК иконку  на панели инструментов и двойным щелчком ЛК по ГЭ открыть его свойства.

Перейти на вкладку кривые ГЭ *Тренд* (рис. 2.120)

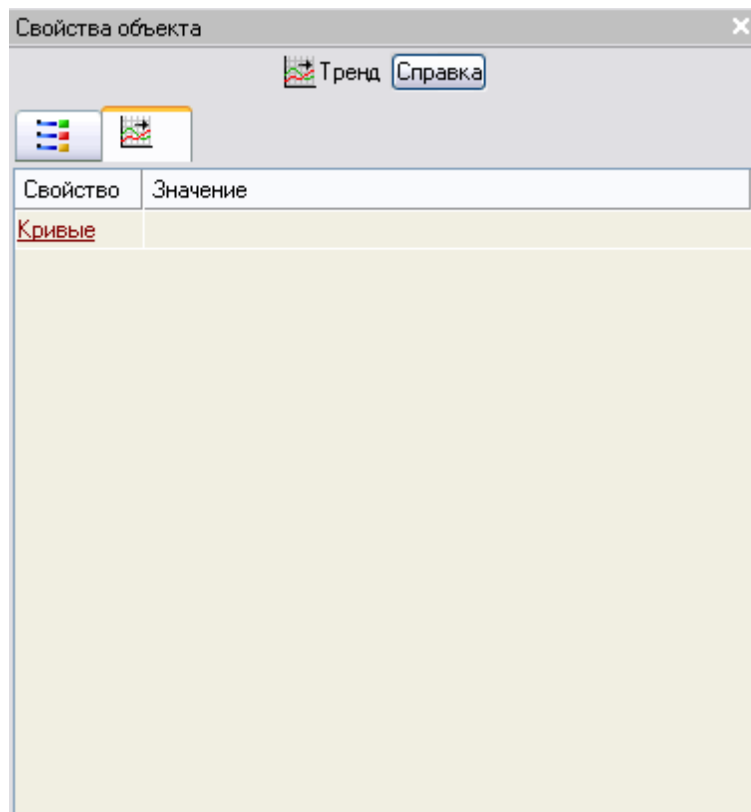


Рисунок 2.120. Свойства ГЭ *Тренд*

Щелкнув ПК по свойству **Кривые** вызвать контекстное меню (рис. 2.121)

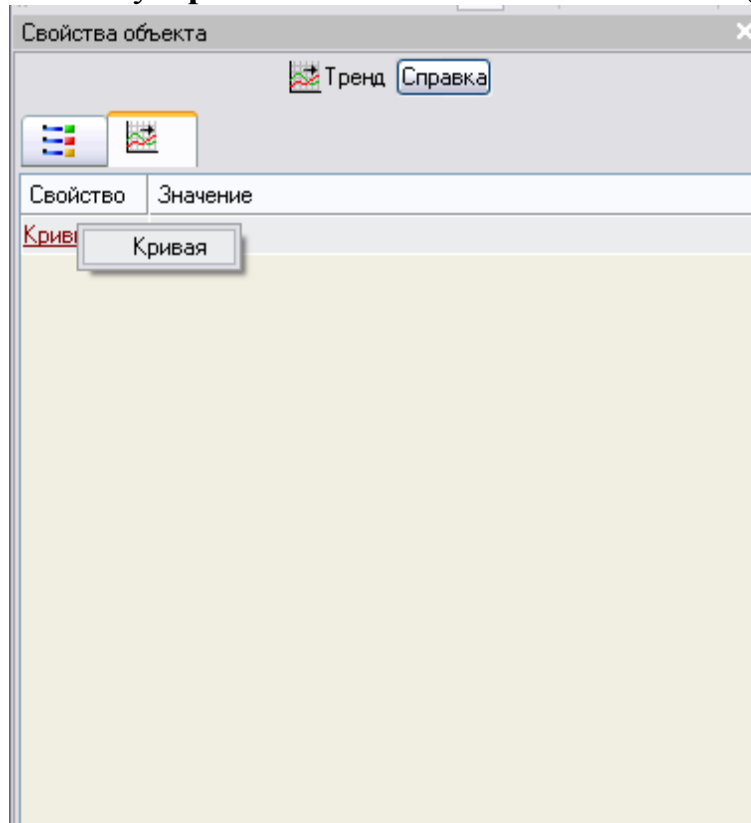


Рисунок 2.121. Свойства ГЭ *Тренд* – создание *Кривой*

Выбрав ЛК команду кривая создать новую кривую для отображения изменения уровня во времени (рис. 2.122).

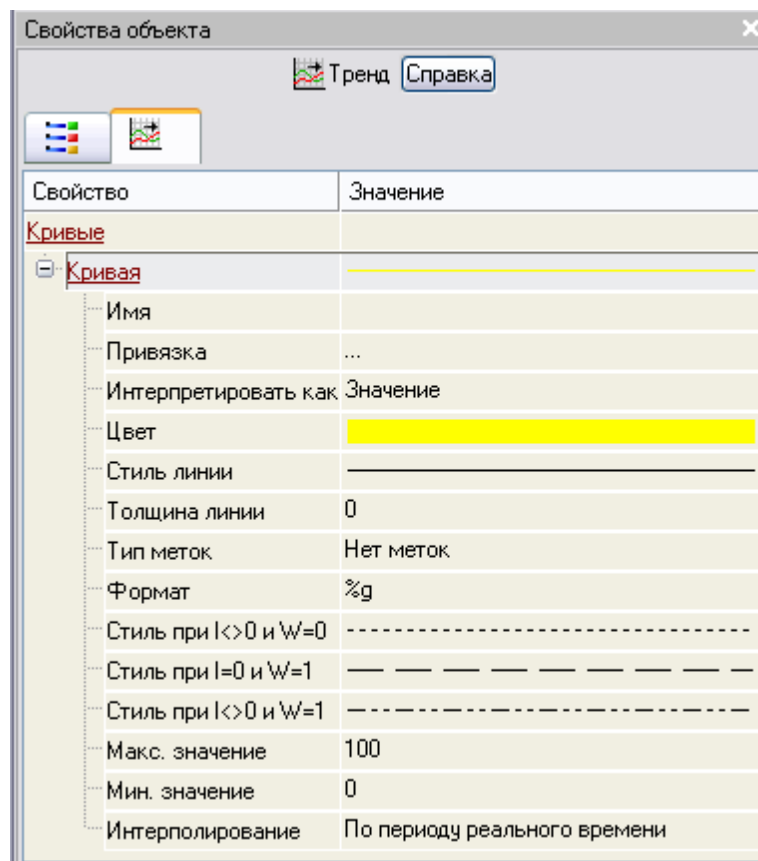


Рисунок 2.122. Свойства *Кривой* ГЭ *Тренд*

Щелчком ЛК по значению *Привязка* открыть окно **Свойства привязки** (рис. 2.123)

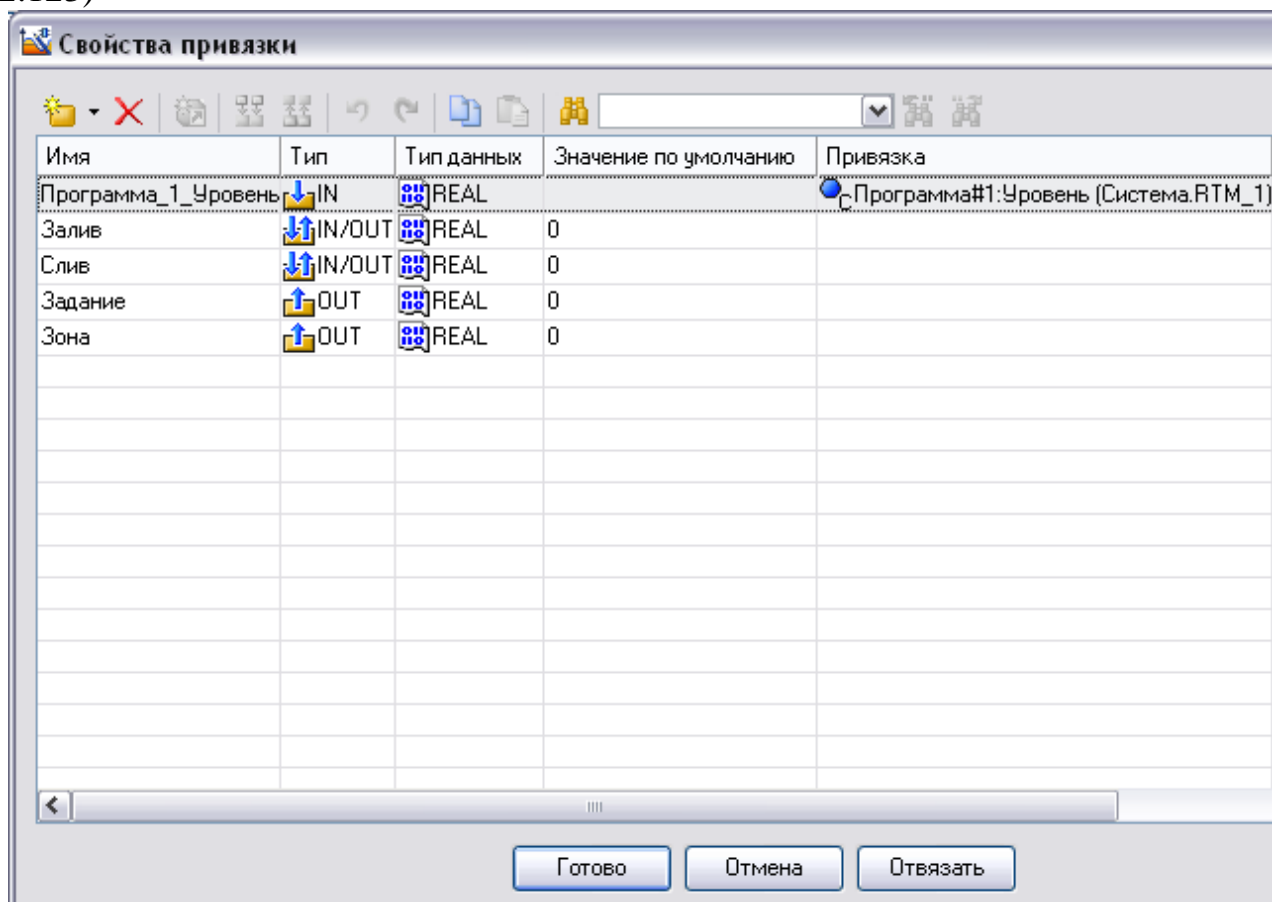
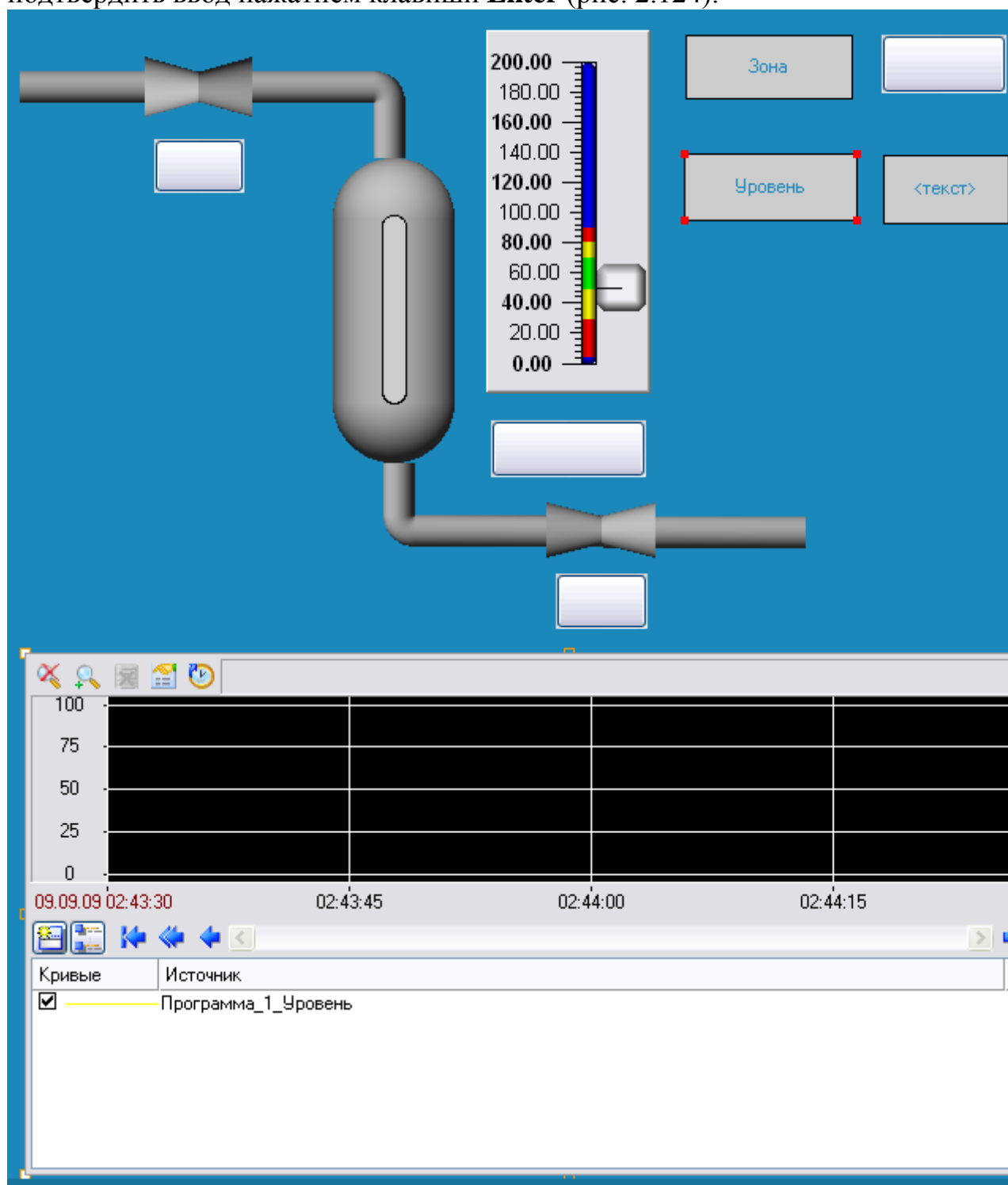


Рисунок 2.123. Свойства привязки

ЛК выбрать аргумент **Программа\_1\_Уровень** и связать его с ГЭ нажав кнопку **Готово**.

Щелчком ЛК в правом поле **Макс. Значение** изменить его на 200 и подтвердить ввод нажатием клавиши **Enter** (рис. 2.124).



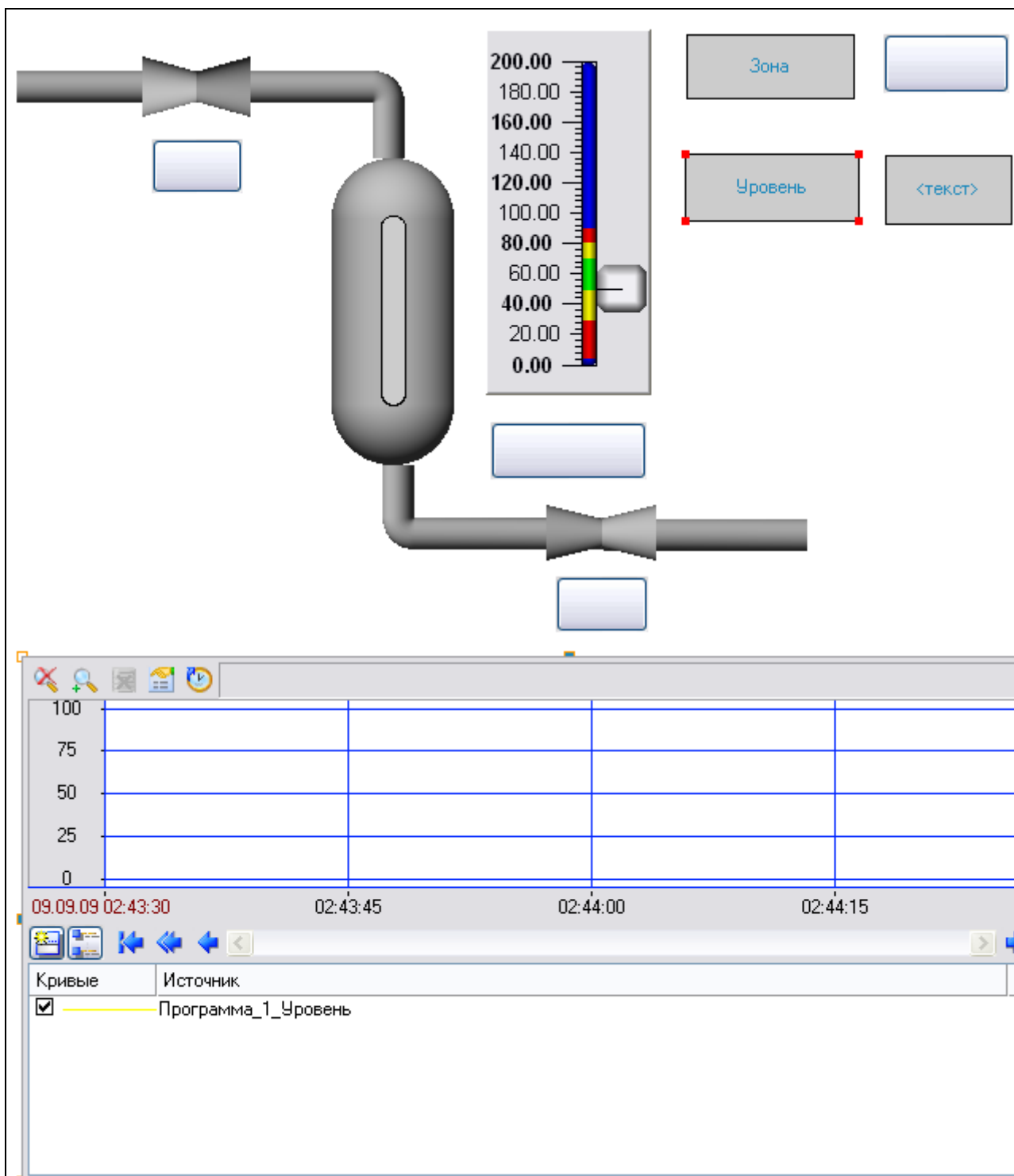


Рисунок 2.124. Изменение ГЭ Тренд

Двойным щелчком ЛК по компоненту **Программа#1** перейдем в режим редактирования программы.

Выделив ЛК в дереве шаблона **Программа#1** строки **Аргументы** вызвать табличный редактор аргументов (рис. 2.125).



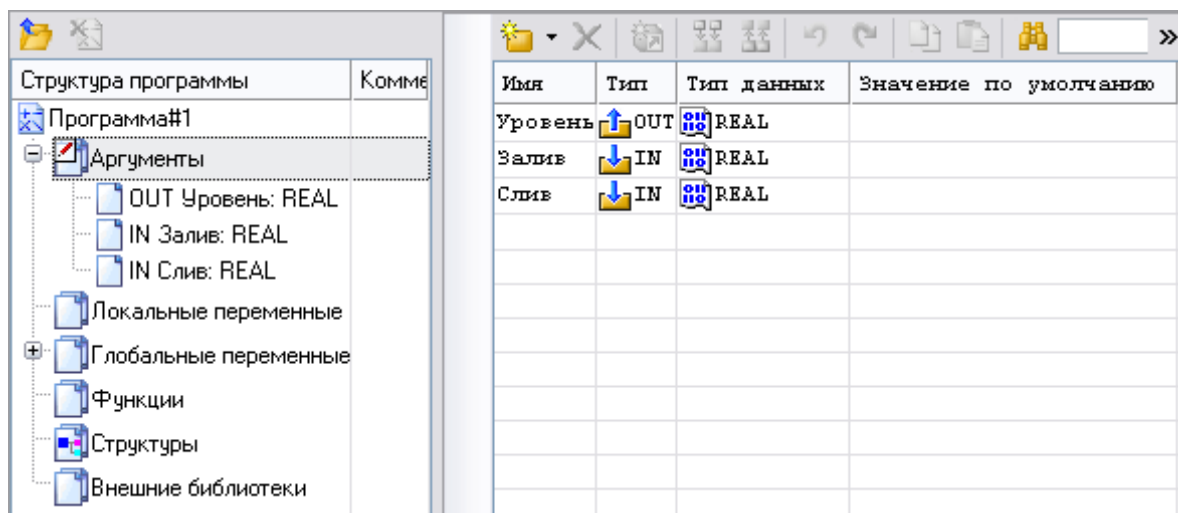


Рисунок 2.125. Привязка аргументов **Программы#1**

Двойным щелчком ЛК по ячейке *Привязка* аргумента **Залив**, вызвать окно конфигурации связей (рис. 2.126).

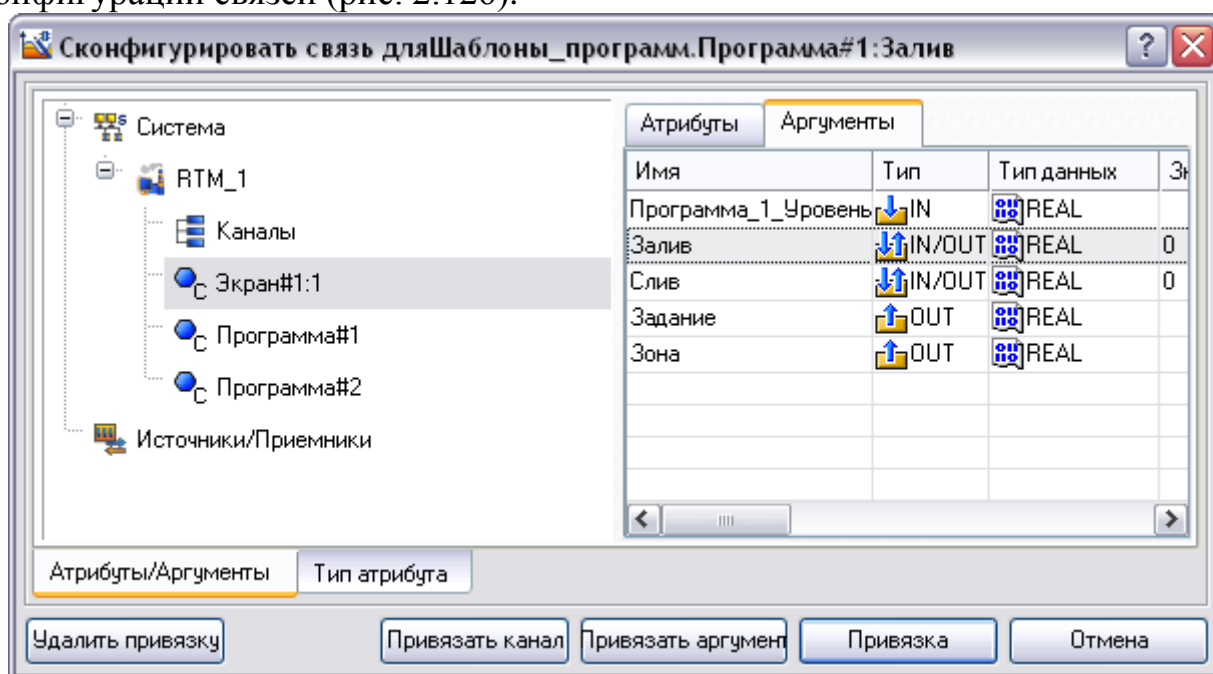


Рисунок 2.126. Привязка аргумента **Залив - Программы#1** с аргументом **Залив - Экрана№1:1**

Выбрать ЛК **Экран№1:1**, переключиться на вкладку аргументы, выбрать ЛК аргумент **Залив** и связать аргумент **Программа#1 - Залив** с аргументом **Экран№1:1 – Залив** нажав кнопку **Привязка**.

Двойным щелчком ЛК по ячейке *Привязка* аргумента **Слив**, вызвать окно конфигурации связей (рис. 2.127).

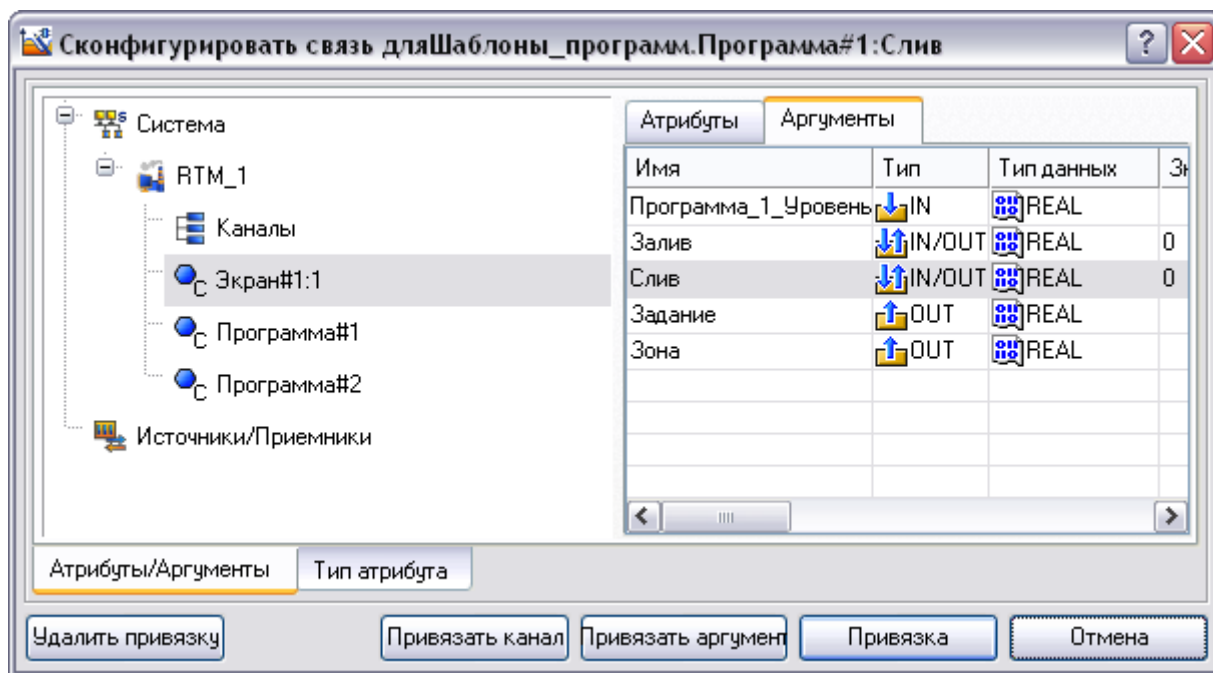


Рисунок 2.127. Привязка аргумента **Слив - Программы#1** с аргументом **Слив - Экрана№1:1**

Выбрать ЛК **Экран№1:1**, переключиться на вкладку аргументы, выбрать ЛК аргумент **Слив** и связать аргумент **Программа#1 - Слив** с аргументом **Экран№1:1 – Слив** нажав кнопку **Привязка**.

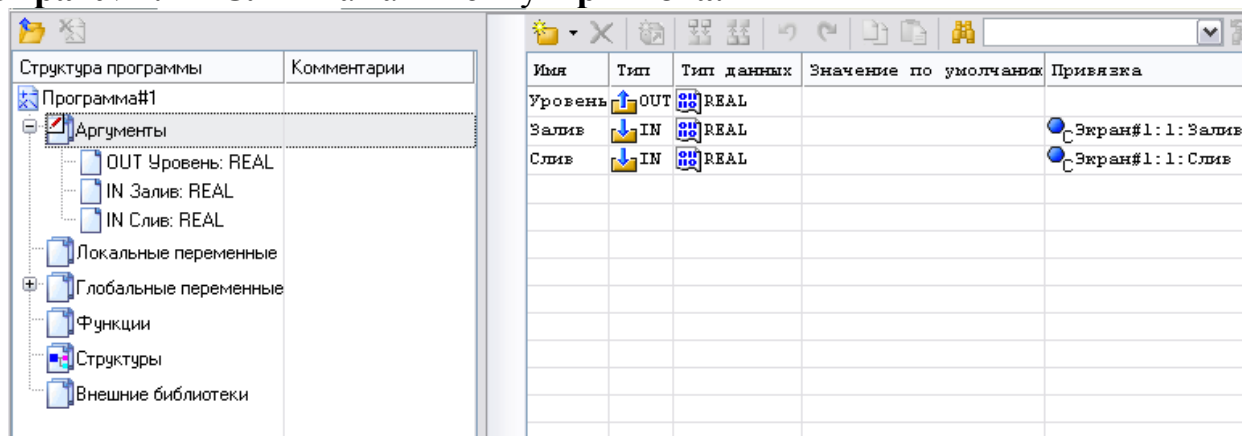


Рисунок 2.128. Привязка аргументов **Программы#1**

**Обратите внимание:** при привязке аргументов их имена могут измениться, в то время как в программе останутся старые имена переменных, в результате чего программа станет не работоспособной. Что бы восстановит работоспособность программы необходимо присвоить аргументам их старые имена.

Двойным щелчком ЛК по компоненту **Программа#2** перейдем в режим редактирования программы.

Выделив ЛК в дереве шаблона **Программа#2** строки **Аргументы** вызвать табличный редактор аргументов (рис. 1.129)

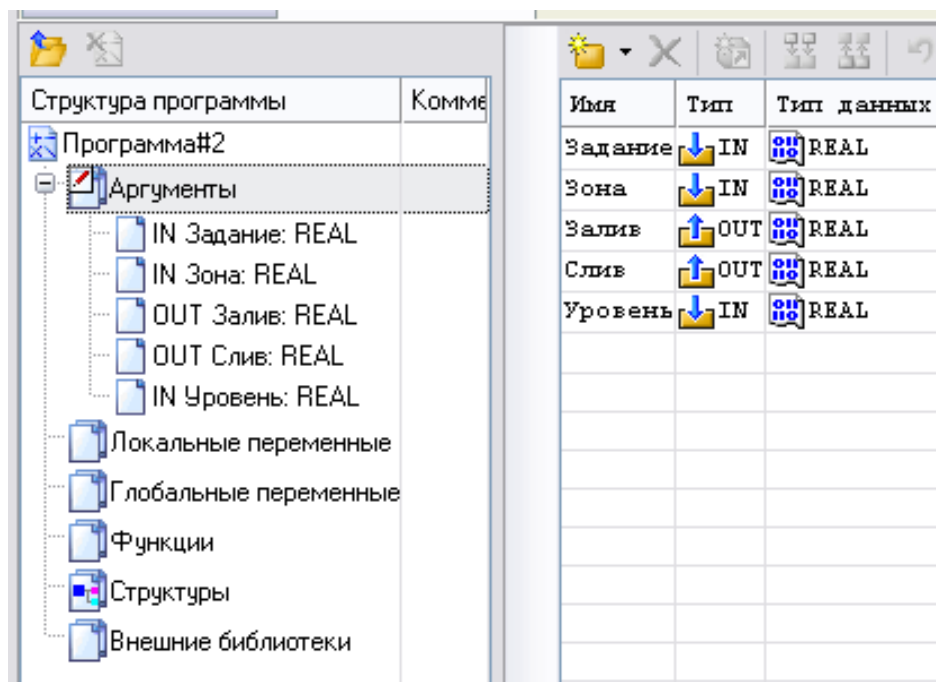


Рисунок 2.129. Аргументы Программы#2

Двойным щелчком ЛК по ячейке *Привязка* аргумента **Задание**, вызвать окно конфигурации связей. Выбрать ЛК **Экран№1:1**, переключиться на вкладку аргументы, выбрать ЛК аргумент **Задание** и связать аргумент **Программа#2 - Задание** с аргументом **Экран№1:1 – Задание** нажав кнопку **Привязка** (рис. 2.130).

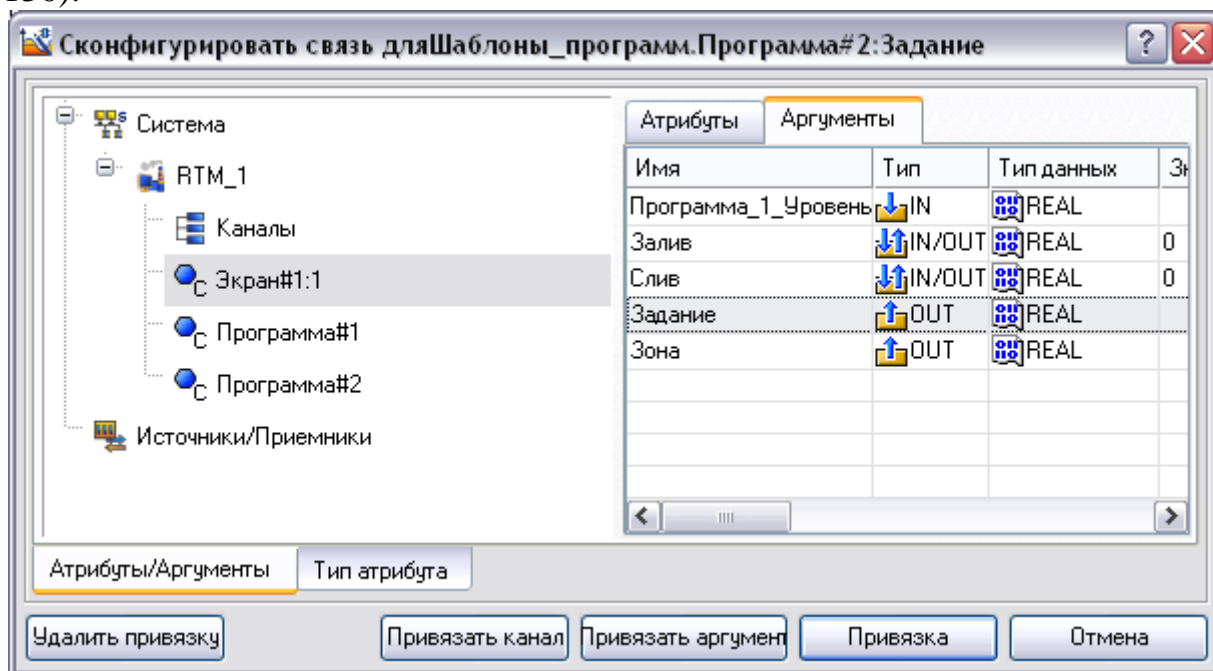


Рисунок 2.130. Привязка аргумента **Задание - Программы#2** с аргументом **Задание - Экрана№1:1**

Двойным щелчком ЛК по ячейке *Привязка* аргумента **Зона**, вызвать окно конфигурации связей. Выбрать ЛК **Экран№1:1**, переключиться на вкладку аргументы, выбрать ЛК аргумент **Зона** и связать аргумент **Программа#2 - Слив** с аргументом **Экран№1:1 – Зона** нажав кнопку **Привязка** (рис. 2.131)

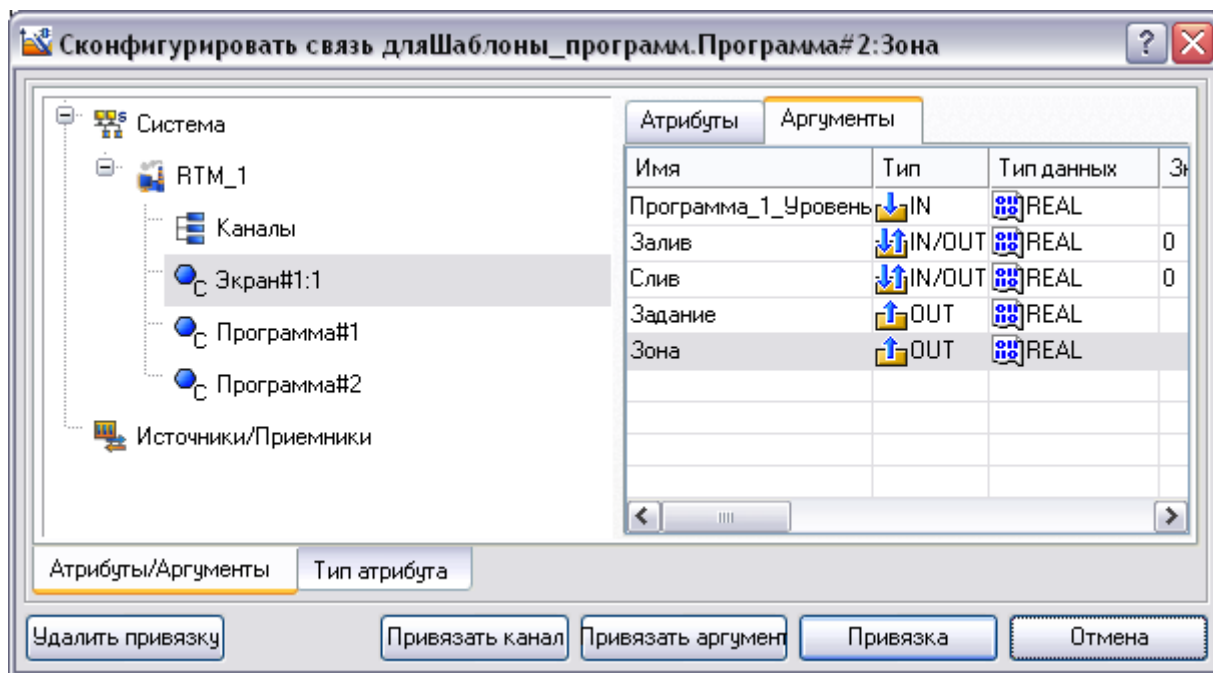


Рисунок 2.131. Привязка аргумента **Зона - Программы#2** с аргументом **Зона - Экрана№1:1**

Двойным щелчком ЛК по ячейке *Привязка* аргумента **Залив**, вызвать окно конфигурации связей. Выбрать ЛК **Экран№1:1**, переключиться на вкладку аргументы, выбрать ЛК аргумент **Залив** и связать аргумент **Программа#2 - Залив** с аргументом **Экран№1:1 – Залив** нажав кнопку **Привязка** (рис. 2.132)

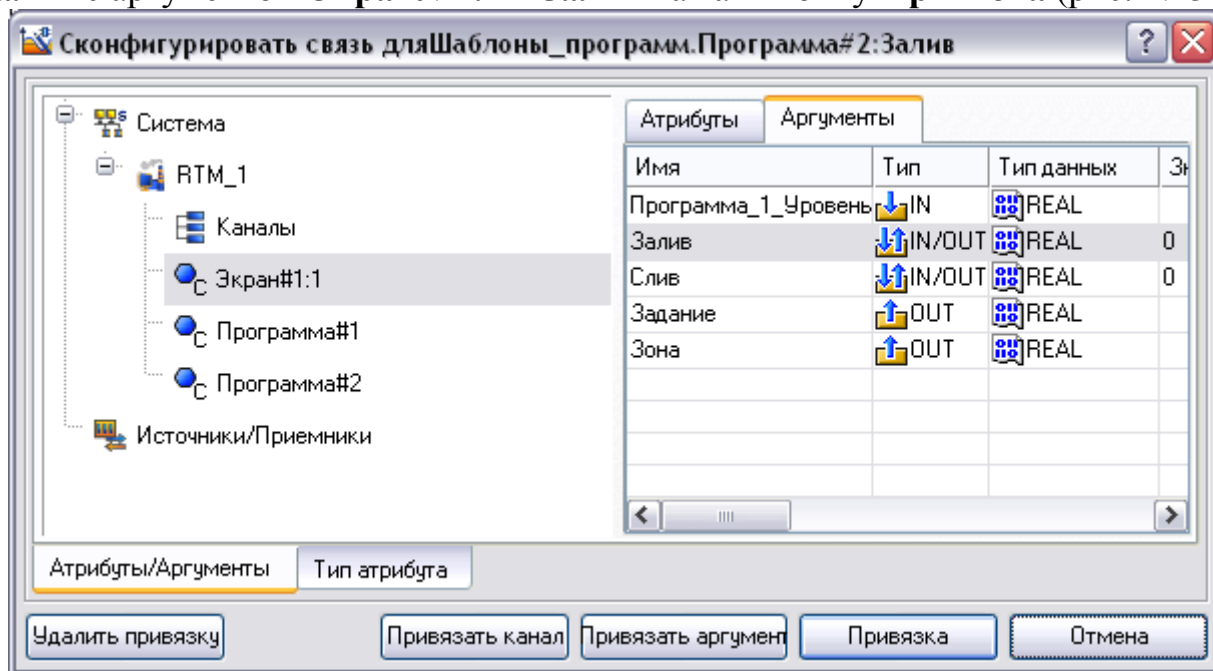


Рисунок 2.132. Привязка аргумента **Залив - Программы#2** с аргументом **Залив - Экрана№1:1**

Двойным щелчком ЛК по ячейке *Привязка* аргумента **Слив**, вызвать окно конфигурации связей. Выбрать ЛК **Экран№1:1**, переключиться на вкладку аргументы, выбрать ЛК аргумент **Слив** и связать аргумент **Программа#2 - Слив** с аргументом **Экран№1:1 – Слив** нажав кнопку **Привязка** (рис. 2.133).

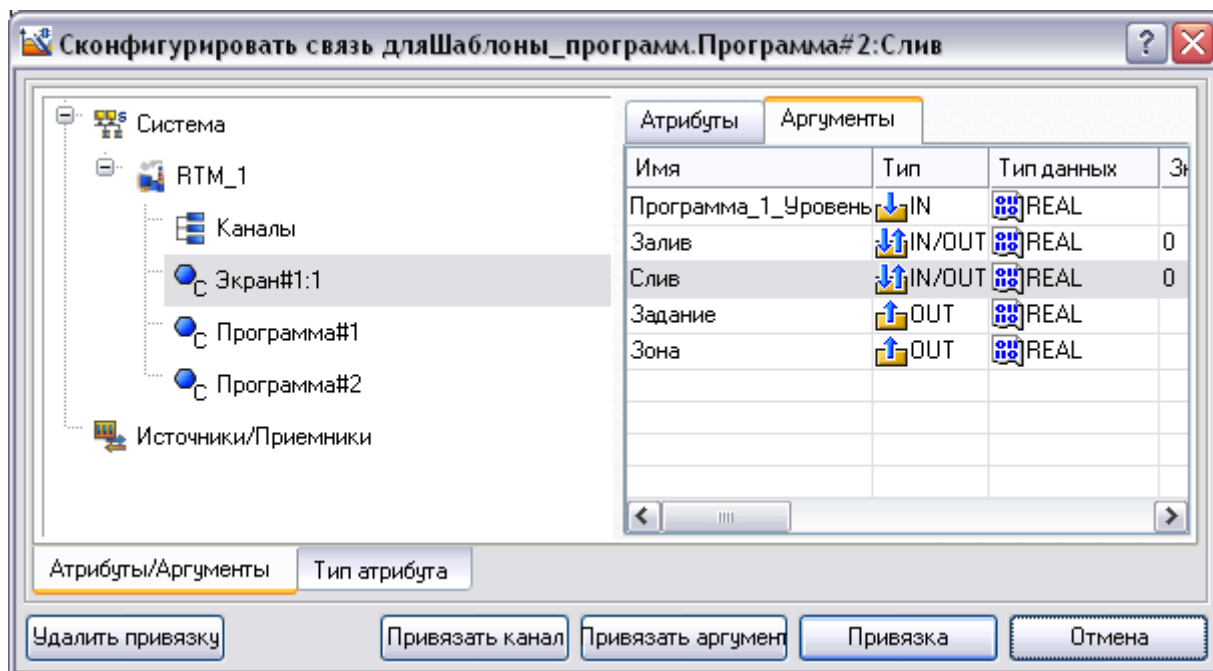


Рисунок 2.133. Привязка аргумента **Слив - Программы#2** с аргументом **Слив - Экрана№1:1**

Двойным щелчком ЛК по ячейке *Привязка* аргумента **Уровень**, вызвать окно конфигурации связей. Выбрать ЛК **Программа#1**, переключиться на вкладку аргументы, выбрать ЛК аргумент **Уровень** и связать аргумент **Программа#2 - Уровень** с аргументом **Программа#1– Уровень** нажав кнопку **Привязка** (рис. 2.134, 2.135).

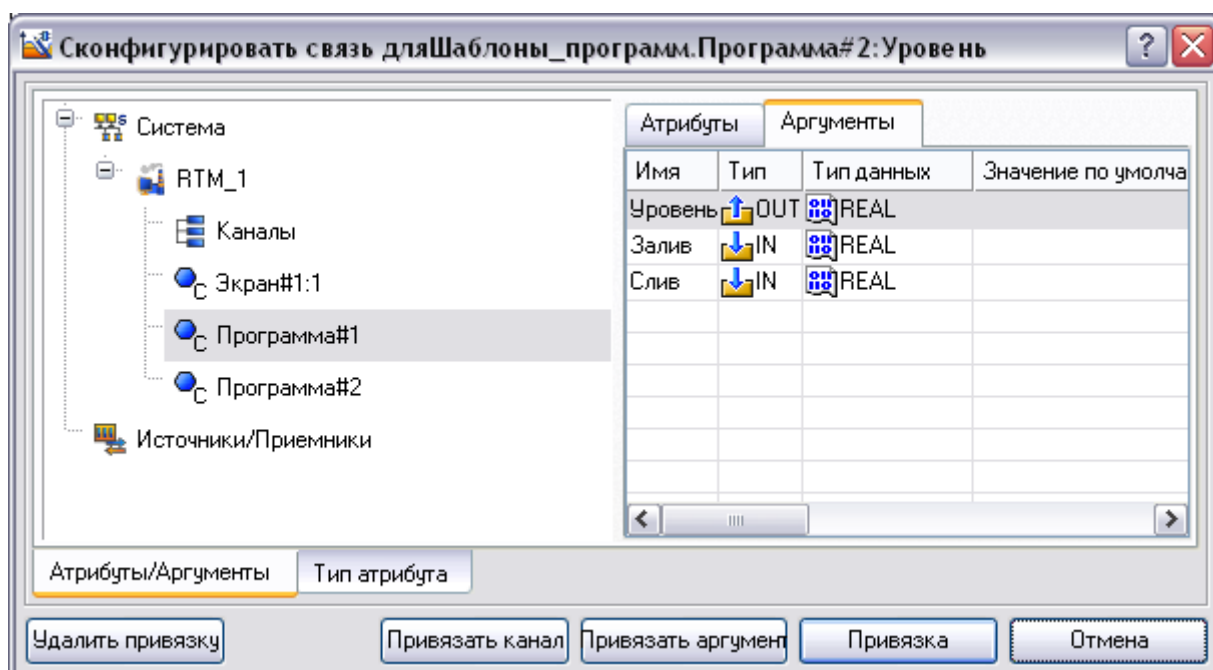


Рисунок 2.134. Привязка аргумента **Уровень - Программы#1** с аргументом **Уровень - Программа#2**

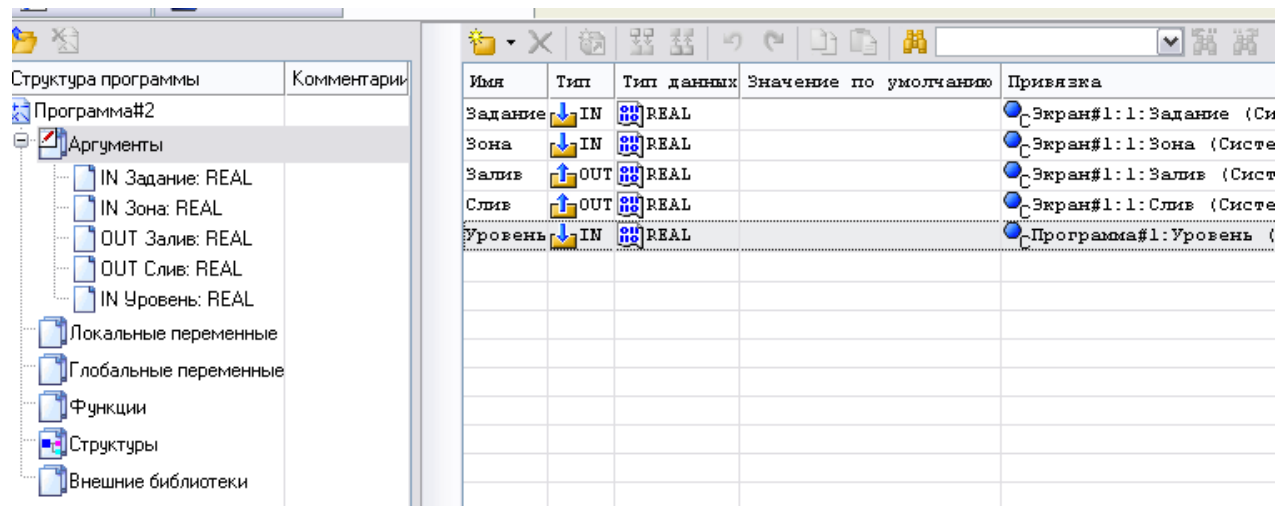




Рисунок 2.135. Аргументы **Программы#2**

Для запуска проекта необходимо нажать иконку  - **Сохранить для MPB.**

Затем  - **Запустить профайлер.**

В открывшемся окне профайлера нажать на иконку  - **Запуск.Останов** (рис. 2.136).

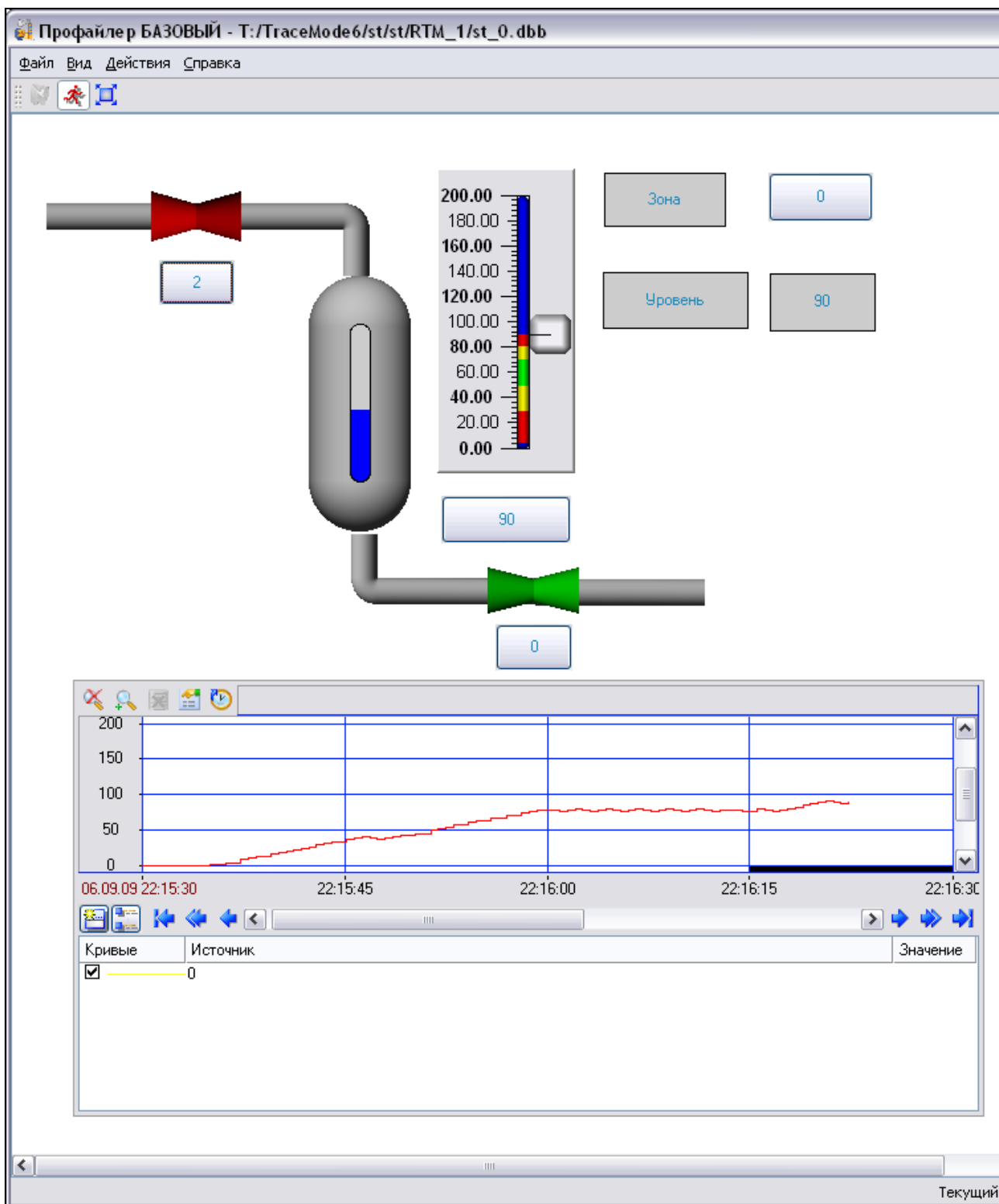


Рисунок 2.136. Работа АСР уровня

Для выполнения работы необходимо снять переходный процесс системы при задании – 90 и зоне нечувствительности – 5. Затем снять скиншот программы комбинацией клавиш **Alt+PrtScn** и вставить его в текстовый редактор комбинацией клавиш **Ctrl+V**.

## СОДЕРЖАНИЕ ОТЧЁТА

Для составления отчета необходимо в меню **Файл** выполнить команду **Документировать проект** полученный \*.html файл необходимо распечатать.

Отчёт должен содержать:

- 1) Титульный лист
- 2) Задание
- 3) Документацию по проекту
- 4) Скриншоты, демонстрирующие работу системы



## ВАРИАНТЫ

№	площадь $S, \text{см}^2$	интервал времени $\Delta t, \text{с}$	максимальный расход поступающей жидкости $F_{\text{max}}^{\text{in}}, \text{см}^3 / \text{с}$	максимальный расход уходящей жидкости $F_{\text{max}}^{\text{out}}, \text{см}^3 / \text{с}$	максимальный уровень ёмкости $L_{\text{max}}, \text{см}$	минимальный уровень ёмкости $L_{\text{min}}, \text{см}$
1	1	1	4	1	300	0
2	1	1	3	2	250	0
3	1	1	2	3	200	0
4	1	1	1	4	150	0
5	1	1	1	1	100	0
6	1	1	2	2	100	0
7	1	1	3	3	150	0
8	1	1	4	4	200	0
9	1	1	4	4	250	0
10	1	1	3	3	300	0
11	2	1	2	2	300	0
12	2	1	1	1	250	0
13	2	1	1	4	200	0
14	2	1	2	3	150	0
15	2	1	3	2	100	0
16	2	1	4	1	100	0
17	2	1	4	1	150	0
18	2	1	3	2	200	0
19	2	1	2	3	250	0
20	2	1	1	4	300	0
21	3	1	1	4	300	0
22	3	1	2	3	250	0
23	3	1	3	2	200	0
24	3	1	4	1	150	0
25	3	1	1	4	100	0

## ВОПРОСЫ ДЛЯ ЗАЩИТЫ

1. Назначение и состав SCADA систем?
2. На каком уровне автоматизации используются SCADA системы.
3. Что такое АРМ оператора.
4. Какие элементы визуализации использовались для создания АРМ оператора.
5. Какие каналы используются для реализации АСР уровня.
6. Дайте краткую характеристику языка LD.
7. Дайте краткую характеристику языка FBD.
8. Дайте краткую характеристику языка SFC.
9. Дайте краткую характеристику языка IL.
10. Дайте краткую характеристику языка CFC.
11. Охарактеризуйте язык структурированного текста ST.
12. Что такое оператор в языке ST?
13. Что такое идентификаторы в языке ST?
14. Что такое ключевые слова в языке ST?
15. Что такое константы в языке ST?
16. Что такое функции в языке ST?
17. Какое утверждение используется в языке ST для завершения утверждения повторения (**for**, **while**, **repeat**) прежде, чем конечное условие будет выполнено?
18. Что такое основная точка входа в программу в языке ST?
19. Что такое массивы в языке ST?
20. Как описываются временные интервалы в языке ST?
21. Тип данных в языке ST?
22. Отличия функции от функции-блока в языке ST?
23. С помощью каких операторов можно построить цикл с убывающим счетчиком в языке ST?
24. Что такое структура в языке ST?

## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Руководство пользователя. TRACE MODE 6 & T-FACTORY. Быстрый старт. Издание пятое (к релизу 6.03.1). Москва 2006. AdAstrA Research Group, Ltd.–163 с.
2. Руководство пользователя. TRACE MODE 6. Издание седьмое (к релизу 6.03.1). Москва 2006. AdAstrA Research Group, Ltd., том 1–554 с. том 2–598 с.
3. Деменков Н.П. SCADA – системы как инструмент проектирования АСУ ТП. М.: Изд-во МГТУ им. Н.Э. Баумана, 2004. – 328 с.
4. Густав Олссон, Джангуидо Пиани. Цифровые системы автоматизации и управления. СПб.: Невский диалект, 2001. – 557с.: ил.

## Приложение 1

### ПРИМЕР ОФОРМЛЕНИЯ ТИТУЛЬНОГО ЛИСТА ЛАБОРАТОРНОЙ РАБОТЫ

Федеральное агентство по образованию

Российский Химико–Технологический Университет  
им. Д. И. Менделеева

Новомосковский институт

Кафедра АПП

Лабораторная работа № 1

по курсу ИСПУ

«Проектирование АСР уровня с использованием языка Structured Text»

студенты	допуск	выполнение	защита

Группа

Преподаватель

Новомосковск 2010 г.

## Приложение 2

### ОПИСАНИЕ TECHNO ST

#### Числовые константы Техно ST

Десятичные целочисленные константы состоят из ненулевой цифры, за которой следует последовательность десятичных цифр:

123, 456, 7890

Двоичные целочисленные константы начинаются с префикса **2#**, за которым следуют цифры 0 или 1:

2#1001, 2#1100

Восьмеричные целочисленные константы начинаются с префикса **8#**, за которым следуют цифры от 0 до 7:

8#777, 8#0123

Шестнадцатеричные константы начинаются с префикса **16#**, за которым следуют цифры или буквы **a...f**. Буквы можно задавать как в нижнем, так и в верхнем регистре (**A...F**):

16#123, 16#EA7

Вещественные константы состоят из целой и дробной части, разделенной точкой. Либо целая, либо дробная часть может отсутствовать. Числа могут задаваться в формате с плавающей точкой, при этом они сопровождаются суффиксом **E** с указанием десятичного порядка:

1.23, 123., .123, 0.123E3, .123e-3, 123.E+5

Временные интервалы состоят из префикса **t#** или **time#**, за которым следует запись в виде

**<дни>d<часы>h<минуты>m<секунды>s<миллисекунды>ms**

Любая составляющая может быть опущена (например, запись **t#1h10s** является корректной и означает 1 час 10 секунд). Временной интервал приводится к целочисленному виду, означающему количество миллисекунд в заданном временном интервале.

Дата состоит из префикса **d#** или **date#**, за которым следует запись в виде **yyyy-mm-dd** (год, месяц, день). Приводится к целочисленному виду, означающему количество секунд, прошедшее с 0 часов 1 января 1970 года до 0 часов заданной даты.

Время дня состоит из префикса **tod#** или **time\_of\_day#**, за которым следует запись в виде **hh:mm:ss** (час, минута, секунда). Приводится к целочисленному виду, означающему количество секунд, прошедшее с 0 часов текущего дня.

Константа "Дата и время" состоит из префикса **dt#** или **date\_and\_time#**, за которым следует запись в виде **yyyy-mm-dd-hh:mm:ss** (год, месяц, день, час, минута, секунда). Приводится к целочисленному виду, означающему количество секунд, прошедшее с 0 часов 1 января 1970 года до заданных даты и времени.

## Идентификаторы Техно ST

Идентификаторы могут состоять из заглавных и строчных латинских букв, цифр и знака подчеркивания '\_'. Первым символом идентификатора не может быть цифра. Длина идентификатора не ограничена.

Идентификаторы не чувствительны к регистру, т.е. 'AAA' и 'aaa' являются идентичными идентификаторами.

Имена констант, переменных, функций и т.п., задаваемые пользователем, должны удовлетворять правилам задания идентификаторов.

## Ключевые слова Техно ST

Ключевые (служебные) слова – это идентификаторы, зарезервированные в языке для специального использования.

Список ключевых слов языка **Техно ST**:

**and, array, bool, break, by, byte, case, constant, continue, date, date\_and\_time, dint, do, dt, dword, else, elsif, end\_case, end\_for, end\_function, end\_function\_block, end\_if, end\_program, end\_repeat, end\_struct, end\_type, end\_var, end\_while, exit, false, for, function, function\_block, goto, handle, if, int, lreal, mod, not, of, or, program, real, repeat, return, rol, ror, shl, shr, sint, string, struct, time, time\_of\_day, to, tod, true, type, uint, until, usint, var, var\_arg, var\_global, var\_inout, var\_input, var\_output, while, word, xor.**

Кроме того, к ключевым словам относятся имена функций C, которые могут быть использованы в ST-программе (см. **Стандартные функции C в ST-программе** ).

Ключевые слова нечувствительны к регистру, т.е. **while** и **WHILE** являются идентичными ключевыми словами.

*Ключевые слова **Техно ST** являются таковыми и для всех других языков; их нельзя использовать в качестве пользовательских идентификаторов (например, в качестве имен переменных) в любых программах.*

## Комментарии Техно ST

Комментарии бывают двух видов: строчные и блочные.

Строчный комментарий начинается с символов // и продолжается до конца строки.

Блочный комментарий начинается с символов /\* и продолжается до символов \*/.

Вложенные комментарии не допускаются.

## ***Лексическая структура языка Техно ST***

В алфавит языка входят:

- ▶ прописные и строчные буквы латинского алфавита;
- ▶ цифры 0,1,...9;
- ▶ специальные знаки:  
+ - \* / < = > ! : & | ^ ~ % ( ) [ ] , ; #

Из символов алфавита формируются лексемы языка:

- ▶ идентификаторы;
- ▶ ключевые слова;
- ▶ числовые и строковые константы;
- ▶ символьные операторы (знаки операций);
- ▶ разделители;
- ▶ комментарии.

### ***Массивы Техно ST***

Массив – это совокупность объектов одного типа, имеющая общее имя (идентификатор). Объекты, составляющие массив, называются элементами этого массива.

Для задания массивов используются операторы определения переменных, для обращения к элементам массивов – оператор индексирования, в связи с чем элементы массивов в **Техно ST** называются также индексированными переменными.

В **Техно ST** определены одномерные и многомерные массивы.

#### **Одномерные массивы**

Одномерный массив – это массив переменных; для обращения к элементу одномерного массива достаточно указания одного индекса. В приведенном ниже примере показаны различные варианты задания одномерных массивов.

#### ***Пример***

PROGRAM

VAR

//Массив 7 переменных типа REAL.

//Элементы массива по умолчанию инициализируются

//нулевыми значениями:

nn: array [7] of REAL;

//Массив 3 переменных типа REAL.

```

//Первые два элемента инициализируются
//заданными значениями
//(n[0]=1.5, n[1] = 10), n[2] по умолчанию
//инициализируется нулевым значением:
    n: array [3] of REAL := 1.5, 2*5;
END_VAR

VAR
//Массив 4 строковых переменных.
//Первые три элемента инициализируются
//заданным значением
//(m[0] = m[1] = m[2] = "OK"), m[3] по умолчанию
//инициализируется пустой строкой:
    m: array [4] of STRING := 3("OK");
END_VAR
//Задание значений элементам массивов:
    n[2] = 7*2.5;
    m[3] = "ERROR";
END_PROGRAM

```

### Многомерные массивы

Многомерный массив – это массив массивов. Определение многомерного массива в общем случае должно содержать сведения о типе, размерности и количествах элементов каждой размерности. Например, трехмерный массив, заданный конструкцией

```
kk: array [4,3,2] of INT;
```

представляет собой массив 4 элементов, каждый из которых – двумерный массив с размерами 3 на 2. Данный массив содержит 24 элемента типа **INT**, которые в приведенном ниже перечне расположены в порядке возрастания адресов (слева направо):

```

kk[0,0,0], kk[0,0,1], kk[0,1,0], kk[0,1,1], kk[0,2,0], kk[0,2,1], kk[1,0,0],
kk[1,0,1], kk[1,1,0], kk[1,1,1], kk[1,2,0], kk[1,2,1], kk[2,0,0], kk[2,0,1],
kk[2,1,0], kk[2,1,1], kk[2,2,0], kk[2,2,1], kk[3,0,0], kk[3,0,1], kk[3,1,0],
kk[3,1,1], kk[3,2,0], kk[3,2,1]

```

### *Пример*

В данном примере показаны различные варианты задания многомерных массивов.

```

PROGRAM
VAR
//двумерный массив переменных
//INT с заданием начальных значений

```



```

//(ll[0,0]=1, ll[0,1]=2, ll[0,2]=3,
// ll[1,0]=4, [1,1]=5, ll[1,2]=6):
    ll: array [2,3] of INT := 1,2,3,4,5,6;
    END_VAR
    VAR
//двумерный массив строковых
//переменных с указанием диапазонов индексов и
//заданием начального значения первых четырех
//элементов
//(pp[5,9]=pp[5,10]=pp[5,11]="OK",
//pp[6,9]= "NO",
//pp[6,10]= ...=pp[7,11]=""):
    pp: array[5 .. 7, 9 .. 11] of STRING:=3("OK"), "NO";
    END_VAR
//задание значений элементов массивов
    ll[1,2] = 17*5;
    pp[6,10] = "ERROR";
END_PROGRAM

```

### Операторы Техно ST

- Оператор return
- Оператор if-then-else
- Оператор case
- Оператор while
- Оператор repeat
- Оператор for
- Операторы break и exit
- Оператор continue
- Операторы определения переменных
- Оператор индексирования элементов массива
- Оператор goto

Определены следующие операторы, образующие предложения **Техно ST**:

**return**

**if**

**case**

**while**

**repeat**

**for**

**break**

**exit**

**continue**

**операторы определения переменных;**

**оператор индексирования элементов массива;**

**goto**

### ***Оператор return***

Определены 2 варианта задания данного оператора:

**return {выражение}**

Действие: выход из функции **Техно ST**. Значением функции является значение {выражения};

**return**

Действие: выход из функции-блока **Техно ST**.

### ***Пример***

RETURN (2 + ARG\_000 \*\* 2);

### ***Оператор if-then-else***

Данный оператор начинается с ключевого слова **if** и заканчивается ключевым словом **end\_if**. Определены 3 варианта задания данного оператора:

#### ***Вариант 1***

**if {выражение} then {последовательность предложений} end\_if**

Действие: если {выражение} истинно, выполняется {последовательность предложений}, иначе никаких действий не производится.

#### ***Вариант 2***

**if {выражение} then {последовательность предложений1}**

**else {последовательность предложений2} end\_if**

Действие: если {выражение1} истинно, выполняется {последовательность предложений1}, иначе выполняется {последовательность предложений2}.

### **Вариант 3**

```
if {выражение1} then {последовательность предложений1}  
elsif {выражение2} then {последовательность предложений2}  
...  
elsif {выражениеN} then {последовательность предложенийN}  
else {последовательность предложений} end_if
```

Действие: выполняется первая по порядку {последовательность предложений}, для которой соответствующее {выражение} истинно. Если все {выражения} ложны, выполняется {последовательность предложений}, следующая за ключевым словом **else**.

Количество блоков "**elsif** {выражение} **then** {последовательность предложений}" не ограничено.

### **Пример**

В результате выполнения следующего кода переменной VAR\_000 присваивается значение 200. Выполняется только одно (первое по порядку) действие, для которого условие истинно, поэтому действие, следующее за конструкцией **ELSIF...THEN**, выполнено не будет, несмотря на то, что условие VAR\_002 < 1 истинно:

```
VAR VAR_000 : INT; END_VAR  
VAR VAR_002 : REAL := 0.5; END_VAR  
IF VAR_002 < 2 THEN  
    VAR_000 = 200;  
ELSIF VAR_002 < 1 THEN  
    VAR_000 = 500;  
ELSE  
    VAR_000 = 300;  
END_IF;
```

### **Оператор case**

Определены 2 варианта задания данного оператора.

### **Вариант 1**

```
case {выражение} of  
    {список значений}: {последовательность предложений}  
    ...  
    {список значений}: {последовательность предложений}  
end_case
```

## **Вариант 2**

**case {выражение} of**

    {список значений}: {последовательность предложений}

    ...

    {список значений}: {последовательность предложений}

**else {последовательность предложений}**

**end\_case**

Список значений представляет собой набор целых чисел или набор диапазонов целых чисел, разделенных запятой. Диапазон задается в виде

**{нижняя граница} .. {верхняя граница}**

Действие: если результат вычисления {выражения} принадлежит множествам, заданным {списками значений}, выполняется соответствующая {последовательность предложений}. Если результат вычисления {выражения} не принадлежит ни одному из заданных множеств, выполняется {последовательность предложений}, следующая за ключевым словом **else**.

### **Пример**

В результате выполнения следующего кода VAR\_001=500:

```
VAR VAR_000 : INT; END_VAR
```

```
VAR VAR_001 : INT; END_VAR
```

```
  CASE VAR_000 + 4 OF
```

```
    0 .. 2 : VAR_001 = 200;
```

```
    3, 4, 5 : VAR_001 = 500;
```

```
  END_CASE;
```

## **Оператор while**

Синтаксис:

**while {выражение} do {последовательность предложений} end\_while**

Действие: пока {выражение} истинно, выполняется {последовательность предложений}.

### **Пример**

После выполнения следующего кода VAR\_001=16:

```
VAR VAR_000 : INT := 10; END_VAR
```

```
VAR VAR_001 : INT; END_VAR
```

```
  WHILE VAR_000 > 2 DO VAR_000 = VAR_000 - 1;
```

```
    VAR_001 = VAR_001 + 2;
```

```
  END_WHILE;
```

## ***Оператор repeat***

Синтаксис:

**repeat {последовательность предложений} until {выражение} end\_repeat**

Действие: пока {выражение} истинно, выполняется {последовательность предложений}. Если {выражение} ложно, {последовательность предложений} выполняется 1 раз.

### ***Пример***

После выполнения следующего кода VAR\_001=20:

```
VAR VAR_000 : INT :=10; END_VAR
```

```
VAR VAR_001 : INT; END_VAR
```

```
REPEAT VAR_001 = VAR_001 + 2; VAR_000 = VAR_000 + 1; UNTIL  
VAR_000 < 20 END_REPEAT;
```

## ***Оператор for***

Синтаксис:

**for {инициализация переменной цикла} to {выражение1} by  
{выражение2} do {последовательность предложений}  
end\_for**

Инициализация переменной цикла имеет вид:

**{имя переменной}:= {выражение}**

Действие: пока значение переменной цикла меньше или равно значению {выражения1} выполняется {последовательность предложений}. По завершении каждого цикла к переменной цикла прибавляется значение {выражения2}; если оно не задано, прибавляется 1.

*С помощью оператора **for** нельзя построить цикл с убывающим счетчиком. Для создания таких циклов нужно использовать операторы **while** и **repeat**.*

### ***Пример***

После выполнения следующего кода VAR\_001=22:

```
VAR VAR_000 : INT :=10; END_VAR
```

```
VAR VAR_001 : INT; END_VAR
```

```
FOR VAR_000 = 10 TO 20 DO VAR_001 = VAR_001 + 2; END_FOR;
```

## ***Операторы break и exit***

Операторы **break** и **exit** эквивалентны.

Синтаксис:

**break**

**exit**

Действие: выход за пределы цикла. В случае вложенных циклов выход осуществляется только из текущего цикла и не затрагивает внешние.

### ***Оператор `continue`***

Синтаксис:

**continue**

Действие: переход в конец цикла, т.е. выражения, следующие за оператором **continue** до конца цикла, не выполняются.

### ***Операторы определения переменных***

Операторы определения переменных могут быть заданы вручную (кроме операторов определения глобальных переменных) или с помощью табличного редактора.

В языке **ST** определены следующие операторы данного типа:

**var**

{определение переменной}

...

{определение переменной}

**end\_var**

**var\_global**

{определение переменной}

...

{определение переменной}

**end\_var**

**var\_arg**

{определение переменной}

...

{определение переменной}

**end\_var**

**var\_input**

{определение переменной}

...

{определение переменной}

**end\_var**

```

var_output
    {определение переменной}
    ...
    {определение переменной}
end_var

```

```

var_inout
    {определение переменной}
    ...
    {определение переменной}
end_var

```

После ключевого слова **end\_var** точка с запятой не ставится.

Действие: определяет новую переменную. При использовании совместно с **constant** задает константу.

Оператор **var ... end\_var** используется для создания локальных переменных и структур; может использоваться в основной программе или ее компоненте (функции).

Оператор **var\_global ... end\_var** используется для создания глобальных переменных; может использоваться вне основной программы и ее компонентов (функций).

Оператор **var\_arg(var\_input) ... end\_var** используется для определения аргументов (основной программы или ее функций), передаваемых по значению. Определение аргумента с помощью этого оператора равнозначно заданию аргумента типа **вход** в табличном редакторе.

Оператор **var\_output(var\_inout) ... end\_var** используется для определения аргументов (основной программы или ее функций), передаваемых по ссылке. Определение аргумента с помощью оператора **var\_output...end\_var** равнозначно заданию в табличном редакторе аргумента типа **выход**, а определение аргумента с помощью оператора **var\_inout...end\_var** равнозначно заданию аргумента типа **вход/выход**.

*Создание аргументов вручную с помощью указанных операторов может использоваться только в отладочных программах – для таких аргументов нельзя задать привязку. Аргументы рабочей программы следует создавать с помощью табличного редактора.*

Выражение {определение переменной} имеет вид:

```

{имя переменной}: {тип переменной};
{имя переменной}: {тип переменной}:={выражение};
{имя переменной}: array [] of {тип переменной};

```

**{имя переменной}: array [{размерности массива}] of {тип переменной};**

**{имя переменной}: array [{размерности массива}] of {тип переменной}:= {начальные значения};**

Выражения {размерности массива} задаются в виде диапазонов изменения индексов массива, разделенных запятой.

Диапазон изменения индексов массива имеет вид

**{нижняя граница} .. {верхняя граница}**

или

**{размер массива}**

обозначающий диапазон от 0 до {размер массива}-1. В случае, если размерность массива не указана, он считается пустым и ожидается его инициализация в ходе выполнения программы.

Выражения {начальные значения} имеют вид списка начальных значений элементов массива, разделенных запятой. Каждое начальное значение имеет вид выражения, вычисление которого дает реальное начальное значение, или конструкции

**{целочисленная константа} ({выражение})**

где {целочисленная константа} задает количество элементов, которым присваивается это значение. При присвоении начальных значений элементам массива первым изменяется последний индекс массива.

Область действия имени переменной определяется по следующим правилам:

- ▶ **глобальные** переменные действуют в рамках программы и сохраняют свое значение между вызовами программы. В частности, глобальными являются переменные FBD- и LD-блоков;
- ▶ **локальные** переменные и **аргументы** действуют в рамках объекта (программы, функции, структуры), в котором определены.

При привязке переменной по имени она ищется в следующем порядке: локальные, аргументы функции, переменные-члены структуры, глобальные.

### ***Оператор индексирования элементов массива***

Синтаксис:

**{имя} [ {индекс 1}, ... {индекс N}]**

где {имя} – имя переменной или функции, возвращающей массив, а {индекс k} – целое неотрицательное число или целочисленная переменная (кроме **UINT** и **USINT**). Количество индексов зависит от размерности массива.

Действие: возвращает ссылку на элемент массива, которая может быть использована в левой и правой части оператора присваивания.



## Оператор goto

Синтаксис:

**goto {метка строки}**

Действие: безусловный переход к строке кода с указанной меткой. Оператор **goto** и метка, на которую этот оператор ссылается, должны находиться в одном и том же программном компоненте (программе, функции и т.п.). Метка должна начинаться с буквы и отделяться от кода программы двоеточием:

```
...
goto myLabel2;
...
myLabel2:
END_PROGRAM
```

## Определение переменных и констант

- Особенности присвоения значений переменным

Вид константы или переменной (глобальная, локальная) задается оператором, с помощью которого данная переменная (константа) определяется (см. **Операторы определения переменных** в разделе **Операторы Техно ST** ). Синтаксис операторов определения переменных предполагает обязательное указание типа данных:

```
//определение локальной строковой
//переменной myVar
VAR myVar: STRING; END_VAR
```

Тип данных определяет размер выделяемой памяти. Для указания типа в **Техно ST** определены следующие ключевые слова (в круглых скобках указано соответствие типу данных C):

**BOOL (bool)** – булево значение размерностью 1 байт (**true (1)** или **false (0)**);  
**SINT (\_\_int8)** – целое со знаком размерностью 1 байт (**-128 ... 127**);  
**USINT (unsigned \_\_int8)** – целое без знака размерностью 1 байт (**0 ... 255**);  
**INT (short)** – целое со знаком размерностью 2 байта (**-32768 ... 32767**);  
**UINT (unsigned short)** – целое без знака размерностью 2 байта (**0 ... 65535**);  
**DINT (long)** – целое со знаком (4 байта) (**-2147483648 ... 2147483647**);  
**UDINT (unsigned long)** – целое без знака (4 байта) (**0 ... 4294967295**);  
**TIME, DATE, TIME\_OF\_DAY, DATE\_AND\_TIME** – соответствуют **DINT**.  
Значения переменных этих типов задаются аналогично соответствующим временным константам (см. **Числовые константы Техно ST** );  
**REAL (float)** – вещественное число (4 байта) (максимальное значение **3.402823466e+38**);

**LREAL (double)** – вещественное число (8 байт) (максимальное значение **1.7976931348623158e+308**);

**STRING (char [])** – 256 символов в кодировке UTF-8 (512 байт, см. также **Строковые константы Техно ST**);

**HANDLE** – специальный тип, используемый для хранения внешних данных в виде числа, имеет размерность 4 байта, не может быть использован в арифметических, логических и т.п. операциях.

Кроме указанных типов, переменной может быть присвоен структурный тип, созданный пользователем. Такая переменная является конкретным объектом указанного типа (см. **Структуры Техно ST**).

При определении переменной может быть задано ее значение:

```
VAR i: INT:=0; END_VAR
```

Если при определении переменной ее значение не задано, то этой переменной по умолчанию присваивается следующее начальное значение:

числовая переменная – 0;

переменная типа **BOOL** – **FALSE**;

переменная типа **STRING** – пустая строка;

переменная типа **HANDLE** – 16#00000000 (0 в формате HEX);

переменная типа **TIME**, **DATE**, **TIME\_OF\_DAY** или **DATE\_AND\_TIME** – 0.

При определении константы задание ее значения обязательно:

```
VAR CONSTANT myConst: INT:=13; END_VAR
```

В отличие от переменной, значение константы в программе изменять нельзя.

### ***Особенности присвоения значений переменным***

При присвоении значения переменной типа TYPE1 переменной типа TYPE2 нужно учитывать следующее:

- ▶ присвоение корректно только в том случае, если тип TYPE2 включает в себе все числа типа TYPE1:

```
VAR a: REAL := -1564.343; END_VAR
```

```
VAR b: USINT := 50; END_VAR
```

```
a = b; //корректная операция
```

```
b = a; //некорректная операция
```

- ▶ присвоение корректно, если один из типов – **BOOL**, а другой – любой численный. Логическое значение TRUE соответствует единице, FALSE – нулю; нуль соответствует FALSE, любое ненулевое значение, в том числе отрицательное, соответствует TRUE:

```
VAR a: BOOL; END_VAR
```

```
VAR b: SINT := -50; END_VAR
```

```
a = b; //a = TRUE, корректная операция
```

```
b = a; //b = 1, корректная операция
```

## Основная точка входа в программу

Основная точка входа в программу определяется следующей конструкцией:

**program**

{определение аргументов}

{список предложений}

**end\_program**

Необязательное выражение {определение аргументов} задается аналогично выражению {определение переменной} для операторов определения переменной (см. раздел **Операторы Техно ST**). В дальнейшем конструкция **program...end\_program** называется основной программой.

Функции, глобальные переменные и структурные типы не могут быть определены в основной программе.

Основная точка входа создается автоматически при создании программы. Если для программы выбран язык **ST** или **IL**, конструкция **program ... end\_program** отображается в листинге. Если для программы выбран язык **SFC**, **LD** или **FBD**, основная точка создается во внутреннем представлении и недоступна для просмотра.

## Особенности вычислений

Целочисленность результата арифметических вычислений в программе имеет высший приоритет – даже в том случае, когда этот результат присваивается переменной с плавающей точкой.

Пусть, например, в программе объявлена переменная **float**:

VAR VAR\_000 : REAL; END\_VAR

Тогда:

VAR\_000 = 2 / 10            //VAR\_000 = 0

VAR\_000 = 2. / 10          //VAR\_000 = 0.2

VAR\_000 = 2. / 10 + 2 / 10 //VAR\_000 = 0.2

## Переменные и константы Техно ST

Под объектом в **Техно ST** понимается некоторая область памяти, которой присвоено имя (идентификатор). Переменная (константа) – это частный случай объекта как именованной области памяти. Отличительной чертой переменной (константы) является возможность связывать с ее именем различные значения, совокупность которых определяется типом переменной (константы).

При определении значения переменной в соответствующую ей область памяти помещается некоторый код. Если этот процесс происходит во время компиляции программы, он называется инициализацией переменной, если во время выполнения программы – присвоением значения.

## Пользовательские функции Техно ST

- Определение функции и функции-блока
- Вызов функции и функции-блока

В **Техно ST** определены две разновидности пользовательских функций: собственно функция и функция-блок. Функция-блок играет роль подпрограммы, т.е. не возвращает значений.

Функции и функции-блоки создаются с помощью табличного редактора. Если при создании указан тип данного компонента, создается функция, в противном случае – функция-блок.

### *Определение функции и функции-блока*

Синтаксис:

```
function {имя} : {тип возвращаемого значения}  
    {определение аргументов}  
    {список предложений}  
end_function
```

```
function_block {имя}  
    {определение аргументов}  
    {список предложений}  
end_function_block
```

Эти конструкции создаются автоматически при создании функции и функции-блока в табличном редакторе.

Необязательное выражение {определение аргументов} задается аналогично выражению {определение переменной} для операторов определения переменной (см. **Операторы определения переменных** в разделе **Операторы Техно ST**). Тело функции составляет {список предложений}.

Если при определении аргумента задано его значение, он считается опциональным. Такой аргумент может не указываться в вызове функции; в этом случае используется значение, заданное аргументу при его описании. После определения опционального аргумента не может следовать определение аргумента, для которого значение не задается (такой аргумент обязательно указывается в вызове функции).

Функция может возвращать значение структурного типа, а также массив.

### *Вызов функции и функции-блока*

Синтаксис:

```
{имя функции}({аргументы})
```

где {аргументы} – последовательность выражений, разделенных запятыми, или

**{имя функции}()**

если аргументы у функции или функции-блока отсутствуют.

*Число аргументов в вызове функции (функции-блока) должно быть равно числу аргументов, заданных для этой функции (функции-блока).*

Передача аргумента по значению или по ссылке задается определением аргумента функции (см. **Операторы определения переменных** в разделе **Операторы Техно ST** ). Строки и массивы всегда передаются по ссылке. В случае, когда соответствующий аргумент определяет передачу по значению, передача все равно осуществляется по ссылке, а аргумент считается константой.

*Термин **по значению** обозначает передачу значения аргумента в вызываемую функцию. При этом функция оперирует с копией переменной и не может изменить ее истинное значение. Термин **по ссылке** обозначает передачу адреса аргумента в вызываемую функцию. При этом функция оперирует с самой переменной и может изменить ее значение.*

Вызываемая функция или функция-блок может быть запрограммирована на любом из встроенных языков (функцию нельзя запрограммировать на **Техно SFC**) – см. также **Создание пользовательских функциональных блоков** .

Следующие функции (функции-блоки) могут быть вызваны в основной программе только однократно:

- ▶ содержащие глобальные переменные программы;
- ▶ содержащие FBD-блоки с внутренними переменными (см. **Редактирование FBD-программ** ).

Другие функции могут вызываться в основной программе многократно.

### **Пример**

Данная программа выполняет возведение в квадрат с использованием ST-функции. При этом показано отличие передачи значения аргумента по ссылке и по значению:

```
FUNCTION SecondDegree: LREAL
//VAR_ARG определяет передачу значения
//переменной loc_var1
    VAR_ARG xx: REAL; END_VAR
//VAR_INOUT определяет передачу адреса
//переменной loc_var2
    VAR_INOUT yy: REAL; END_VAR
//следующее действие не изменяет
//значение loc_var1
    xx = xx+1;
//следующее действие изменяет значение loc_var2
    yy = yy+1;
```

```

    RETURN (xx+yy)**2;
END_FUNCTION
PROGRAM
    VAR
        loc_var1:INT;
        loc_var2:INT;
        d : LREAL;
    END_VAR
    loc_var1 = 2;
    loc_var2=1;
    // передача loc_var1 и loc_var2 в функцию,
    // способ передачи задается в функции.
    // Переменной d присваивается значение,
    // определенное в функции оператором RETURN
    d := SecondDegree(loc_var1, loc_var2);
END_PROGRAM

```

### ***Пример***

Данный пример демонстрирует особенности использования функции-блока.

```

FUNCTION_BLOCK Sec_Degree
    VAR_ARG
        xx: REAL;
        yy: REAL;
    END_VAR
    VAR_OUTPUT
        result: LREAL;
    END_VAR
    result =(xx+yy+1)**2;
END_FUNCTION_BLOCK
PROGRAM
    VAR
        loc_var1:INT:=2;
        loc_var2:INT:=1;
        d : LREAL;
    END_VAR
    //Передача значений loc_var1 и loc_var2
    //в функцию-блок. Переменной d присваивается
    //значение переменной result функции-блока.
    //Чтобы такое присвоение работало корректно,
    //переменная result должна быть определена в

```

```
//функции-блоке оператором VAR_OUTPUT или
//VAR_INOUT
    Sec_Degree(loc_var1, loc_var2, d);
END_PROGRAM
```

## Разделители Техно ST

В качестве разделителей, или знаков пунктуации, в языке **Техно ST** используются следующие лексемы:

```
+ - * ** / < <= <> << > >= >> ! != = == : := & | ^ ~ % ( ) [ ] . ..
, ;
```

## Синтаксис Техно ST

Для описания структуры программы и операторов в **Техно ST** приняты следующие терминологические соглашения:

- ▶ **выражение** – последовательность операндов, разделителей и символьных операторов, задающая вычисление без присвоения результата;
- ▶ **предложение** – последовательность лексем, определяющая выполнение логически законченного промежуточного действия. Таким действием может быть присвоение переменной результата вычислений, вызов функции-блока и т.п. Операторы (кроме символьных) также образуют предложения.

На основании этих соглашений программа или ее компонент на языке **Техно ST** определяется как последовательность предложений.

Каждое предложение должно завершаться точкой с запятой. Исключением из этого правила являются операторы определения переменных, для завершения которых точка с запятой не используется.

Длина строки программы не ограничивается, лексемы разделяются произвольным числом пробелов, знаков табуляции или символов перевода строки.

## Специальные функции в ST-программе

В ST-программе могут быть использованы следующие специальные функции:

```
//чтение байта из порта с номером port_num
    USINT inp(UINT port_num)
//чтение слова из порта с номером port_num
    UINT inpw(UINT port_num)
//запись байта в порт с номером port_num
```

```

    outp(UINT port_num, USINT value)
//запись слова в порт с номером port_num
    outpw(UINT port_num, UINT value)
//чтение атрибута канала (целое со знаком,
//4 байта)
    DINT getAttributeI(UDINT ch_id, UINT attr_id)
//чтение атрибута канала (вещественное, 4 байта)
    REAL getAttributeF(UDINT ch_id, UINT attr_id)
//установка атрибута канала (целое со знаком,
//4 байта)
    setAttributeI(UDINT ch_id, UINT attr_id, DINT value)
//установка атрибута канала (вещественное,
//4 байта)
    setAttributeF(UDINT ch_id, UINT attr_id, REAL value)

```

*Функции чтения и записи в порт поддерживаются только в Микро MPB для DOS.*

В качестве **ch\_id** может выступать число, равное ID канала, или аргумент, привязанный к атрибуту 118, **ID** канала.

В качестве **attr\_id** и **value** могут выступать числа или аргументы с соответствующими числовыми значениями.

### Стандартные функции C в ST-программе

В ST-программе может быть использован ряд стандартных функций Си:

**sin(), cos(), tan(), asin(), exp(), log()**

### Строковые константы Техно ST

Строковые константы представляют собой набор символов, заключенных в одинарные или двойные кавычки: **'первая строка'**, **"вторая строка"**. В строке недопустимы управляющие символы, включая переводы строки, а также кавычки и символ \$.

Для размещения в строках произвольных символов применяется механизм эскейп-последовательностей, начинающихся с символа \$. Определены следующие последовательности:

- \$r** – возврат каретки, код 16#0D;
- \$n** – перевод строки, код 16#0A;
- \$t** – табуляция, код 16#09;
- \$uXXXX** – UNICODE-символ ('X' – шестнадцатеричная цифра);
- \$x** – символ x ('x' – любой символ).



### Пример

"Строка с кавычкой: \$', символом \$u0410 и переводом строки \$n"

## Структуры Техно ST

В отличие от массива, структура – это совокупность объектов, имеющих различный тип.

Для создания структур нужно вначале с помощью табличных редакторов определить структурные типы, а также переменные, аргументы и функции, которые являются членами этих структурных типов.

**Для справки:** во внутреннем представлении языка **Техно ST** определение структурного типа задается следующей конструкцией:

```
type
  {имя_структурного_типа} : STRUCT
    //определения переменных –
    //членов структуры
    //(см. "Операторы определения
    //переменных")
    //определения функций –
    //членов структуры
    //(см. "Пользовательские функции
    //Техно ST")
  end_struct;
end_type
```

Параметр {имя\_структурного\_типа}, а также переменные и функции – члены структурного типа – задаются с помощью табличных редакторов.

Определив структурный тип, можно определять конкретные структуры данного типа (объекты). Такими объектами могут быть переменные или функции.

Объекты создаются с помощью табличных редакторов. Для этого в основной программе или ее компоненте создается переменная (функция) типа {имя\_структурного\_типа}.

Для обращения к элементам объекта используется так называемое уточненное имя:

{имя объекта}.{имя элемента}

### Пример

Пусть в программе создан структурный тип с именем **myType**, членами которого являются следующие переменные и функция:

```
mvar1: REAL:=3.14;
mvar2: STRING := "OK";
```

```

FUNCTION SecondDegree: LREAL
  VAR_ARG xx: REAL; END_VAR
  RETURN xx**2;
END_FUNCTION

```

Создание и использование объекта типа **myType** иллюстрирует следующий код:

```

PROGRAM
  VAR
    d: LREAL;
    myS : myType; //объект myS типа myType
  END_VAR
  d=myS.SecondDegree(25); //возвращает 625
  d=myS.mvar1; //возвращает 3.14
  //изменение значения mvar1 объекта myS
  myS.mvar1 = 10;
END_PROGRAM

```

Таким образом, структурный тип является только шаблоном структур. С именем типа не связан никакой конкретный объект, поэтому это имя нельзя использовать, например, для формирования уточненного имени элемента:

```

//ошибочная конструкция
myType.mvar1

```

# Проектирование АСР уровня с использованием языка Structured Text