

ГЛАВА

9

Глава 9 Программа "Конструктор OPC-модели"

9.1 Введение

OPC-модель используется при создании автоматизированных рабочих мест (АРМ) на компьютере в среде Windows.

ПО "SyTrack-TOOL" OPCModelBuilder (программа "Конструктор OPC-модели") позволяет создавать модели. На исполнение модель запускается в программе "WinDecont". Для создания программ отображения данных для оператора/диспетчера в компании ДЭП используются Borland C++Builder 6/ CodeGear 2007/ CodeGear 2009/ CodeGear 2010, если они установлены на компьютере, и специальный набор OPC-компонентов и редактор-OPC. Для разработки модели и программы отображения пользователь должен быть знаком с языком программирования "С". Модель и программы отображения могут работать на разных компьютерах, более того, с одной моделью могут работать несколько программ визуализации, и в одной программе отображения можно обрабатывать данные от разных моделей.

Модель является полноценным OPC-сервером (стандарт OPC Data Access 3.0). Работу модели можно просматривать через OPC-проводник, в том числе через программу "DEP OPC Проводник". Дерево элементов для модели создается в программе "Конструктор OPC-модели". Элементами модели могут быть дискретные, аналоги и счетчики контроллера WinDecont, а так же различные расчетные элементы. Модель умеет архивировать свои элементы и вести записи в оперативный журнал. Она может читать и писать данные других OPC-серверов. Модель обеспечивает регистрацию пользователей и ограничение их доступа к элементам модели в зависимости от категории пользователя.

Также, кроме описания структуры объекта, с помощью программы "Конструктор OPC-модели" можно создавать алгоритмы управления объектом, алгоритмы архивирования данных, расчетные алгоритмы, тревоги и предупреждения.



Модель можно использовать в любых других пакетах для автоматизации, поддерживающих OPC-интерфейс данной версии. В комплексе ДЕКОНТ OPC-сервером является и контроллер WinDecont. Поэтому при использовании стороннего ПО для автоматизации, поддерживающего OPC-интерфейс, необходимо оценить, какой OPC-сервер комплекса ДЕКОНТ удобен для решения ваших задач. Напомним, что контроллер WinDecont в качестве OPC-элементов публикует текущие базы параметров: дискретные, аналоги и счетчики.



В ПО "SyTrack-TOOL" OPCModelBuilder предусмотрено лицензирование по количеству вновь создаваемых пользователем проектов:

- до 10 проектов
- от 10 проектов

9.2 Элементы и функции модели

9.2.1 Введение

ПО "SyTrack-TOOL" OPCModelBuilder (программа "Конструктор OPC-модели") предназначено для создания модели - древовидной структуры элементов, которая "видна" через OPC-интерфейс. Такая структура внутри себя может содержать также элементы, невидимые через OPC-интерфейс. В дальнейшем элементы, видимые через OPC интерфейс (узловые элементы), называются просто элементами модели.

Модель имеет тактовый принцип работы. Такт - это время, через которое обновляются значения элементов модели.

Модель использует библиотеку типов и функций. В поставляемой с "Конструктором OPC-модели" библиотекой определены различные типы, которые можно использовать при построении модели. Кроме того, "Конструктор" позволяет создавать свои типы.

При описании элементов в данной документации используется следующая форма:

- **имя**(тип) - означает имя элемента и тип. Имя выделяется **жирным шрифтом**, если оно вводится впервые.

В программе "Конструктор OPC-модели" можно также создавать алгоритмы управления объектом, алгоритмы архивирования данных, расчётные алгоритмы, тревоги и предупреждения.

9.2.2 Базовые элементы

Каждый элемент модели:

- имеет **имя**, уникальное для уровня дерева на котором элемент находится. Имя должно подчиняться правилам для имен языка C, то есть не должно содержать русских букв, не должно начинаться с цифры, не должно содержать специальных символов;
- принадлежит некоторому **типу**. Тип определяет тип значения, которое содержит элемент и другие характеристики элемента;
- может иметь **значение** (логическое, целочисленное, с плавающей точкой, строка, время). Кроме значения, элемент содержит качество данного значения, которое включает признак достоверности этого значения, признак тревожности ("Внимание");
- имеет **название**. Название - это строковое пояснительное описание элемента, оно используется для формирования архивного имени элемента. Название, как правило, русское;
- может быть **замаскирован**. Такой элемент существует в модели, но его не "видно" OPC-клиентам и он не участвует в обработке;
- может быть **сохраняемым**, то есть сохранять свое значение при рестарте модели.

В библиотеке введены следующие **базовые типы**:

Базовый тип	Тип значения
iVoid	нет значения
iBool	логическое
iInt	целое
iFloat	число с плавающей точкой
iString	строка
iDateTime	дата и время
iWord	целое беззнаковое 2-х байтовое значение
iDWord	целое беззнаковое 4-х байтовое значение
iDouble	число с плавающей точкой двойной точности

Кроме базовых существуют **составные** типы. Такие типы содержат элементы (подэлементы) других базовых и составных типов.

Каждый новый тип должен быть **пронаследован** от базового типа или типа, который имеет в цепочке своих родителей базовый тип. Наследование одного типа от другого означает, что наследник получает весь набор подэлементов, характеристики и тип значения родителя. И в дальнейшем можно в новый тип добавлять другие подэлементы и изменять характеристики.

Фундаментальные типы.

Элемент фундаментального типа имеет имя и значение. Он не виден OPC-клиентам. У него нет дополнительных возможностей базовых типов. К фундаментальным типам относят: **void, bool, int, WORD, DWORD, short, BYTE, float, double, AnsiString, TDateTime**, где AnsiString и TDateTime типы из библиотеки VCL фирмы Borland. Элементы фундаментальных типов имеют меньше возможностей, но используют меньше памяти.

Для базового типа можно задать **таблицу расшифровки**, которая задает соответствие целому значению строку. Таблица расшифровки имеет смысл для целочисленных элементов. Таблица используется при архивировании событий, точнее при их просмотре. Если элемент имеет целочисленное значение и для него задана таблица расшифровки и OPC-клиент запрашивает значение элемента как строку, то OPC-сервер модели вернет строку, указанную в таблице расшифровки соответствующую текущему целому значению элемента. Если такое целое значение в таблице не найдено, то возвратится целое значение, преобразованное в строку.

Более подробно о базовых элементах см раздел [Программирование](#), [Программирование\Пары](#) и [Программирование\Базовые элементы](#).

Единицы измерения.

Элементы работы с базами Windecont: wdDIn, wdDOut, wdDOOutImp, wdAin, wdAOut, wdCIn, wdCOut а также уставки имеют поле Units(String) - единицы измерения. Этот элемент выступает в качестве OPC-свойства "Единицы измерения" (Номер свойства-100). Если элемент архивируется, то Units соответствуют единицам измерения архивного элемента.

OPC-свойства

Базовый элемент может быть OPC-свойством элемента, в котором он находится.

Для этого его **номер свойства** устанавливается не равным 0.

Если номер свойства установлен равным 101 (по стандарту OPC DA этому номеру соответствует описание элемента), то значение этого элемента записывается в название элемента, содержащего данный.

9.2.3 Ссылки, списки, массивы

Ссылка tLink.

Ссылка или псевдоним – это узел модели, значение, достоверность и время которого берутся от другого элемента, на который ссылка указывает. Ссылка имеет те же возможности что и базовый тип. Наследуется от tBase, то есть у нее есть название, ее можно замаскировать, архивировать, она видна OPC-клиентам и т.д. Например, для типа iInt ссылка обозначается как **tLink<iInt>**. Ссылка может указывать только на элемент базового типа.

Массив

Массив – это индексируемая последовательность элементов одного типа. Массив может быть только одномерным.

Индексация начинается с 1. Элементы массива именованы как 'ArrayN', где 'Array' – имя массива, а N – индекс элемента. Размер массива определяется во время редактирования модели. Например, массив из 5 элементов типа `Int` обозначается как **tArray**<`iInt`,5>. Массив может содержать элементы только базового типа. Массив не является элементом модели, то есть не является узлом модели. Его элементы принадлежат элементу модели, в котором массив расположен.

Список

Список – это индексируемая последовательность ссылок [tLink](#) на один тип. Список может быть только одномерным. Индексация начинается с 1. Имя ссылки – ее порядковый номер. В отличие от массива, список является узлом модели и наследуется от типа `iVoid`. Для заполнения списка ему можно указать следующие параметры:

- начальный элемент
- типа заполнения
- **KindName** (`AnsiString`) - имя подэлемента
- **KindStr** (`AnsiString`) - признак принадлежности

При заполнении списка будут рассматриваться элементы, находящиеся в иерархии ниже начального элемента.

Если начальный элемент не задан, то начальным считается элемент, в котором находится список.

Если тип заполнения не задан, то считается что тип заполнения `tBase`.

Если не задан начальный элемент и тип заполнения, то список не заполняется.

Элемент может попасть в список, если его тип равен или наследуется от заданного типа заполнения.

Тип заполнения, должен наследоваться или быть равным типу, на который ссылаются элементы списка.

Если задан `KindName`, то элемент попадает в список, если он имеет подэлемент с именем `KindName` и значение его, взятое как строка соответствует `KindStr`.

Полученная строка и `KindStr` имеют следующий формат:

"str1,str2,...,strn".

Значение подэлемента соответствует `KindStr`, `KindStr` - пустая строка или если некоторый `strk` существует и в `KindStr` и в значении подэлемента.

Названия элементов списка заполняются полными названиями элементов, на которые они ссылаются.

Уставки

В понятие уставки входят текущее значение и новое значение. Уставка - это элемент некоторого типа, содержащий новое значение. Значение самой уставки - текущее значение. Имена типов-уставок начинаются с префикса "s". Например, для типа `iInt` - уставка `sInt`, для типа `iFloat` - уставка `sFloat`, то есть для типа `iXXX` - уставка `sXXX`. Уставка `sXXX` наследуется от типа `iXXX` и имеет следующие поля :

- **NewValue**(`iXXX`) - новое значение. По старту работы модели инициализируется значением самой уставки.
- **Control**(`iInt`) - управление.
 - При записи 0 значение уставки копируется в новое значение.
 - При записи 1 новое значение копируется в значение уставки.
- **Units** (`iString`) - [Единицы измерения](#).

Более подробно о [ссылках](#), [массивах](#), [списках](#).

9.2.4 Элементы Windecont

Обычно модель работает с базой текущих значений контроллера WinDecont. Именно так в модель поступают данные из технологической сети (в виде дискретов, аналогов и счетчиков). И через запись в базу текущих значений модель выдает различные управляющие воздействия в сеть контроллеров. Работать можно с контроллером WinDecont, запущенном на одном компьютере с моделью, а также с контроллерами WinDecont, работающими на [других компьютерах](#). Более того одна модель может одновременно работать с несколькими контроллерами WinDecont.

Для работы с базой текущих значений используются типы из группы "Элементы баз WD", условно их можно разделить на

следующие группы:

- доступ к дискретам:
 - wdDIn - чтение дискретов;
 - wdDOut - запись дискрета;
 - wdOutImp - запись импульса в дискрет;
- доступ к аналогам:
 - wdAIn - чтение аналогов;
 - wdAOut - запись аналогов;
- доступ к счетчикам:
 - wdCIn - чтение счетчиков;
 - wdCOut - запись счетчиков;
- индикатор состояния контроллера WinDecont: [wdState](#);
- элементы для расчета относительного номера параметра: [wdNoBase](#). Для каждого элемента доступа к параметру указывается номер параметра в базе WinDecont. В самом простом варианте используется абсолютная нумерация, т.е. указывается номер в базе параметров. Но обычно раскладка параметров имеет регулярную структуру, и в таком случае эффективно использовать относительную нумерацию. При описании типа некоторого объекта задаются относительные номера параметров, а при применении типа (конкретный экземпляр) указывается базовый номер, с которого начинаются параметры данного экземпляра.
- Элементы доступа к базам WD [наследуются](#) от следующих [базовых типов](#):
 - дискреты - от iInt (значение типа int)
 - аналоги - от iFloat (значение типа float)
 - счетчики - от iDWord (значение типа DWORD)так что элементы доступа к базам WD имеют поля и функции этих базовых типов.

Доступ к дискретам

Чтение дискрета осуществляется типом **wdDIn**. wdDIn имеет следующие поля:

- **No** (wdNoD) - номер элемента в базе.
- **Err** (iWord) - код ошибки.
- **ErrStr** (iString) - текст ошибки в зависимости от кода Err.
- **Inv** (bool) - производится ли инверсия значения при чтении из базы. По умолчанию false.
- **Type** (int) - тип обработки бита динамики (по умолчанию 0).
- **Units** (iString) - [Единицы измерения](#).

Для типа wdDIn инверсия работает так: если значение больше нуля - то его инверсия равна 0, иначе 1.

В зависимости от Type, значение wdDIn формируется следующим образом.

- Если Type=0, то производится чтение дискрета (DiscretRead); полученное значение инвертируется, если Inv=true;
- Если Type=1, то производится взятие дискрета и результат получается по следующим правилам:
 - если старое и новое значения достоверны и равны, причем равны 0 или 1, и установлен бит динамики, то результат - инверсия нового значения.
 - иначе результат - новое значение.Такая обработка бита динамики в первом случае обеспечивает формирование импульса, по крайней мере, на один такт. Если Inv = true, то полученное по вышеизложенным правилам значение инвертируется.
- Если Type=2, то производится взятие дискрета (DiscretGet). Type=2 отличается от Type=0 только тем, что в первом случае дискрет читается, а во втором забирается, то есть если установлен бит динамики, то он сбрасывается в базе WD.

Запись дискрета реализуется классом **wdDOut**. wdDOut имеет следующие поля:

- **No** (wdNoD) - Номер элемента в базе.
- **Err** (iWord) - Код ошибки.
- **ErrStr** (iString) - Текст ошибки в зависимости от кода Err.
- **Type** (int) - Тип записи. Может принимать два значения:
 - Type=0 установка дискрета (по умолчанию)
 - Type=1 запись дискрета.
- **Inv** (bool) - Инверсия.
- **Units** (iString) - [Единицы измерения](#).

Для типа wdDOut инверсия работает только для значений 0 и 1.

Если в элемент wdDOut было записано значение или запустился контроллер Windecont, то в зависимости от поля Type в дискрет записывается или устанавливается значение wdDOut.

Если Inv - true, то перед записью, если значение wdDOut равно 0 или 1, то оно инвертируется.

Если значение wdDOut недостоверно, то в Windecont записывается код ошибки, заданный в поле Err.

Установка дискрета отличается от записи тем, что при установке, если значение в базе равно 0, а устанавливаемое значение больше 0,

в дискрете автоматически устанавливается бит динамики.

Запись импульса осуществляется классом **wdDOutImp**. wdDOutImp имеет следующие поля:

- **No** (wdNoD) - Номер элемента в базе.
- **Err** (iWord) - Код ошибки.
- **ErrStr** (iString) - Текст ошибки в зависимости от кода Err.
- **Duration** (float) - Длительность импульса в секундах (по умолчанию 1).
- **Units** (iString) - [Единицы измерения](#).

После записи в wdDOutImp в дискрет записывается с битом динамики некоторое значение, называемое длительностью импульса. Длительность задается в миллисекундах. Длительность импульса вычисляется так: равна значению wdDOutImp, если Duration=0 или Duration*1000, если Duration не равно 0. Если полученная длительность не больше 8191, то в дискрет записывается эта длительность с битом динамики. Если длительность больше 8191, то она переводится в секунды (делится на 1000). Если полученная длительность в секундах больше 8191, то она считается равной 8191. Полученное значение записывается в дискрет с указанием того, что значение выражается в секундах.

Доступ к аналогам

Чтение аналога осуществляется типом **wdAIn**.

Тип wdAIn содержит следующие поля:

- **No**(wdNoD) - Номер элемента в базе.
- **Err**(iWord) - Код ошибки.
- **ErrStr**(iString) - Текст ошибки в зависимости от кода Err.
- **Type**(int) - тип пересчета. По умолчанию 1 – линейный пересчет.
- **X1,X2,Y1,Y2**(float) - граничные условия. По умолчанию X1=Y1=0, X2=Y2=1.
- **Units** (iString) - [Единицы измерения](#).

Тип wdAIn читает аналог из базы, его значение пересчитывается в соответствии с типом пересчета и заносится в значение wdAIn.

Существуют следующие типы пересчетов:

- 0 - нет пересчета
- 1 - линейный $a*x$
- 2 - квадратный корень \sqrt{x}
- 3 - квадрат $x*x$
- 4 - синус $\sin(x)$
- 5 - косинус $\cos(x)$
- 6 - степень x^{10}
- 7 - обратное $1/x$

Пересчет производится по следующим формулам:

линейный:

$$V=KMUL*C+KADD$$

квадрат, квадратный корень:

$$V=KMUL*F(C-CMIN)+KADD.$$

синус, косинус:

$$V = KMUL * F((C - CMIN) / (C MAX - CMIN) * (PI / 2)) + KADD.$$

степень:

$$V = VMIN * ((VMAX / VMIN) ** ((C - CMIN) / (C MAX - CMIN))).$$

обратный:

$$V = KOMUL / C + KOADD.$$

где

V - физическая величина

C - электрическая величина

F - функция

$$KMUL = (VMAX - VMIN) / (C MAX - CMIN).$$

$$KADD = VMIN - KMUL * CMIN.$$

$$KOMUL = C MAX * CMIN (VMAX - VMIN) / (C MAX - CMIN).$$

$$KOADD = VMIN - KOMUL / C MAX.$$

$$VMAX = Y2$$

$$VMIN = Y1$$

$$C MAX = X2$$

$$CMIN = X1$$

Запись аналога с пересчетом осуществляется типом **wdAOut**. Тип содержит следующие поля:

- **No**(wdNoD) - Номер элемента в базе.
- **Err**(iWord) - Код ошибки.
- **ErrStr**(iString) - Текст ошибки в зависимости от кода Err.
- **Type**(int) - тип пересчета. По умолчанию 1 – линейный пересчет.
- **X1, X2, Y1, Y2**(float) - граничные условия. По умолчанию X1=Y1=0, X2=Y2=1.

Если в wdAOut было записано значение или стартовал контроллер Windecont, то оно пересчитывается по параметрам пересчета и записывается в Windecont.

Если значение wdAOut недостоверно, то в Windecont записывается код ошибки Err.

Доступ к счетчикам

Чтение счетчика осуществляется классом **wdCIn**. wdCIn содержит следующие поля:

- **No**(wdNoD) - Номер элемента в базе.
- **Err**(iWord) - Код ошибки.
- **ErrStr**(iString) - Текст ошибки в зависимости от кода Err.
- **Km**(float) - коэффициент умножения. По умолчанию 1.
- **Kd**(float) - коэффициент деления. По умолчанию 1.
- **Delta**(iDWord) - Изменение значения счетчика за 1 такт
- **Type**(int) - Тип чтения, по умолчанию 0.
- **DecCor**(bool) - Десятичная коррекция.
- **Units** (iString) - [Единицы измерения](#).

Тип wdCIn читает значение счетчика с номером No из базы WinDecont. Если прочитанное значение достоверно, то оно умножается на Km/Kd и заносится в значение wdCIn. Delta устанавливается равной разности полученного значения и текущего значения wdCIn. Если установлена десятичная коррекция и прочитанное значение меньше значения wdCIn, то Delta находится с учетом десятичной коррекции. Если прочитанное значение недостоверно, то

- если тип чтения 0, то значение Delta и wdCIn устанавливаются недостоверными;
- если тип чтения 1, то значение wdCIn не меняется, а Delta устанавливается равным 0.

Запись счетчика осуществляется типом **wdCOut**. wdCOut содержит следующие поля:

- **No**(wdNoD) - Номер элемента в базе.
- **Err**(iWord) - Код ошибки.
- **ErrStr**(iString) -Текст ошибки в зависимости от кода Err.
- **Km**(float) - коэффициент умножения. По умолчанию 1.
- **Kd**(float) - коэффициент деления. По умолчанию 1.
- **Delta**(iDWord) - Изменение счетчика.
- **Units** (iString) - [Единицы измерения](#).

При записи в wdCOut или старте контроллера Windecont в базу WinDecont записывается значение wdCOut умноженное на Km/Kd. Если значение wdCOut недостоверно, то в базу записывается код ошибки Err.

Запись в Delta увеличивает значение самого элемента wdCOut и счетчика в базе Windecont на Delta.

Расчет номера параметра

В библиотеке существуют типы **wdNoD**, **wdNoA** и **wdNoC**, которые определяют номера дискретов, аналогов и счетчиков. Эти типы наследуются от базового типа iWord, содержат в себе целое число - номер дискрета, аналога или счетчика. Для удобства установки номеров введен тип **wdNoBase** - номероноситель. Номероноситель наследуется от iVoid и содержит три подэлемента (имя(тип)):

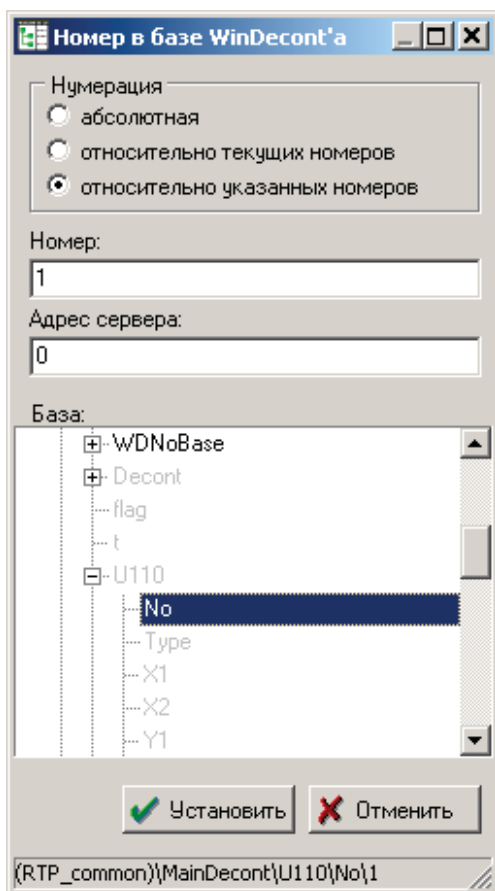
- **BaseD** (wdNoD) - базовый номер для дискретов.
- **BaseA** (wdNoA) - базовый номер для аналогов.
- **BaseC** (wdNoC) - базовый номер для счетчиков.

При настройке wdNoX (wdNoD, wdNoA, wdNoC) можно указать не только абсолютный номер в базе, но и указать относительный номер и элемент wdNoBase, относительно которого будет подсчитываться номер wdNoX. Есть три возможности настройки номера wdNoX:

- **абсолютная нумерация**: указан абсолютный номер. В этом случае во время работы модели значение wdNoX равно указанному номеру.
- **нумерация относительно текущих номеров** : указан относительный номер. Ищется текущий элемент wdNoBase, это первый элемент wdNoBase, находящийся "выше" в структуре модели относительно данного wdNoX. Значение wdNoX равно сумме значения соответствующего поля wdNoX, найденного элемента wdNoBase, и указанного относительного номера.
- **нумерация относительно указанных номеров**: указан относительный номер и элемент wdNoBase, относительно которого будет подсчитано значение wdNoX. Указанный элемент wdNoBase должен быть расположен "выше" в структуре модели, чем wdNoX. При подсчете значения wdNoX будет взят указанный элемент wdNoBase, его соответствующее поле, и значение этого поля добавлено к указанному относительному номеру. Полученное значение - значение wdNoX.

В номере параметра есть поле OpSrvID (Адрес сервера), которое указывает, с каким контроллером WinDecont будет связан данный элемент.

Пример заполнения номера в программе "Конструктор модели".



Состояние работы контроллера WinDecont

Тип **wdState** показывает состояние работы локального контроллера Windecont. Он наследуется от типа **iBool**. Значение элемента данного типа равно **true**, если контроллер Windecont работает и значения всех элементов типа **wdNoA**, **wdNoD**, **wdNoC** находятся в допустимых пределах.

Для индикации работы удаленных контроллеров WinDecont используйте поле **State** в элементе, описывающем данный [удаленный сервер](#).

Некоторые вопросы по программированию см в разделе [Программирование\Элементы Windecont](#).

9.2.5 Тревоги

Для элементов модели можно определить тревожное состояние. Для одних это будет превышение некоторого порогового

значения, для других это установка значения в некоторое predetermined и так далее. В результате тревожного состояния в элементе может быть установлен признак "Внимание", который доступен через OPC-интерфейс (в битах качества OPC-элемента), и просмотреть его можно, например, в программе "ДЭП OPC Проводник". Он сохраняется вместе со значением элемента (для сохраняемых элементов). Обычно в программах отображения тревожное состояние элемента обозначается миганием.

В библиотеке модели предлагается объединять элементы с одинаковым способом определения тревожного состояния в группы, которые управляются сводным элементом. Тип сводного элемента соответствует классу тревоги. Все сводные элементы имеют следующие поля:

- **List** - список контролируемых элементов. Для заполнения списка надо указать два параметра:
 - TypeName - имя типа контролируемых элементов;
 - Root - начальный узел, внутри которого будут найдены элементы указанного типа;
- **AutoReset**(bool) - автоматический сброс значения элемента тревоги в 0, если все контролируемые элементы находятся в неаварийном состоянии;
- **AlarmControl**(bool) - устанавливать признак "Внимание" тревоги у контролируемого элемента;
- **Control** (iInt) - управление значением и состоянием тревоги. Данное поле не заполняется в "Конструкторе OPC-модели", запись в данное поле при работе модели осуществляется квитирование тревоги.

Сводный элемент, как и любой другой элемент модели, имеет значение и признак "Внимание". Значение элемента тревоги изначально равно 0. При переходе контролируемого элемента в тревожное состояние значение сводного элемента устанавливается в 1 и устанавливается признак "Внимание". Установка поля **AutoReset** приводит к возвращению значения сводного элемента в 0 после перехода всех контролируемых элементов в нетревожное состояние.

При установленном поле **AlarmControl** признак "Внимание" взводится также в контролируемом элементе. Сбрасывается признак "Внимание" тревоги только через запись в подэлемент **Control**.

В элемент **Control** можно записать любую комбинацию «битового ИЛИ» 1,2,4:

- 1 – установить значение элемента тревоги равным 0, что возможно при нахождении всех контролируемых элементов в нетревожном состоянии.
- 2 – сбросить признак "Внимание" элемента тревоги;
- 4 – сбросить признак "Внимание" у всех контролируемых элементов.

Далее приводится описание различных типов тревоги, они различаются способом определения аварийного состояния контролируемого элемента.

aToBad - тревожное состояние - это недоверенное значение контролируемого элемента.

aValue - тревожное состояние определяется признаком "Внимание" контролируемого элемента, который взводится при изменении значения, не считая недоверенного. Для сводного элемента AlarmControl игнорируется, и работа происходит как при установленном AlarmControl.

aIState - контролируемый элемент в аварийном состоянии, если его значение достоверно и удовлетворяет условиям CheckedState. Данная тревога имеет поле

- **CheckedState**(AnsiString); Проверяемое состояние. По умолчанию 0.

CheckedState имеет следующий формат:

знак_a1;знак_a2;...знак_aN, где знак_ - одно из:

- пустая строка - то же самое что и знак "="
- "=" - равенство
- "!=" - неравенство
- ">=" - больше или равно
- "<=" - меньше или равно
- ">" - больше
- "<" - меньше

a1,..,aN - числа.

Значение контролируемого элемента удовлетворяет CheckedState, если выполняется одно из условий в CheckedState.

Например, если в CheckedState строка "3;=5;>=10", то элемент находится в аварийном состоянии, если его значение равно 3, 5 или больше или равно 10.

Обратите внимание, что тип aIState не работает с контролируемыми элементами типа iWord или типа наследующегося от

iWord.

aiAlarm - контролируемый элемент в тревожном состоянии при установленном признаке "Внимание". При использовании тревог данного типа бит "Внимание" устанавливаются (и возможно сбрасываются) в тактовой функции контролируемого элемента (см. раздел "Программирование"). Поле AlarmControl игнорируется.

aiSetPoints - контролируемый элемент находится в аварийном состоянии, если его значение достоверно и находится вне пределов Hi и Lo:

- float **Hi,Lo**; Верхний и нижний пределы. По умолчанию 0.

aiSPNames - для каждого контролируемого элемента находятся входящие в него подэлементы с именами Hi и Lo, значения которых используются как пределы, вне которых считается, что контролируемый элемент находится в аварийном состоянии.

- AnsiString **Hi,Lo**; Имена ограничивающих элементов.

Некоторые вопросы по программированию см. в разделе [Программирование\Тревоги](#).

Пример использования тревоги

Рассмотрим общие идеи формирования тревоги на примере контроля состояния оборудования. В базе дискретов формируется текущее состояние связи с некоторым контроллером или модулем ввода/вывода: 1 - связь есть, не определено - связи нет. В АРМе обычно отображается текущее состояние связи с каждым контроллером. Есть сводный сигнал для КП: норма - есть связь со всеми контроллерами КП, авария - нет связи хотя бы с одним из контроллеров данного КП. Сводный сигнал обычно отображается на обобщенных мнемосхемах. Для внимания оператора важен факт пропадания связи с ранее работавшими контроллерами. При обнаружении такого события рекомендуется включить звуковую сигнализацию и световой сигнализацией (обычно миганием) указать в каком КП произошла неисправность и с каким именно контроллером. Т.е. сводный индикатор состояния оборудования КП и индикатор состояния контроллера должны мигать. Мигание и звуковая сигнализация снимаются "Квитированием" данного события оператором.

Для реализации данного алгоритма требуется сделать следующее. Предположим у нас есть тип TCp, который содержит все элементы некоторого КП:

- определяем новый тип SModule, наследуя его от wdDIn;
- для всех дискретов состояния связи в TCp меняем тип с wdDIn на SModule. Для вновь создаваемого TCp просто добавляем элементы типа SModule;
- добавляем в TCp элемент тревоги AlarmModule тип aiToBad;
 - List
 - Root = (TCpType)\
 - TypeName = SModule
 - AutoReset = true (когда связь установится, тревога снимется автоматически)
 - AlarmControl = true (что бы взводился бит тревоги в контролируемом элементе - тогда на мнемосхеме можно мигать модулями, которые перестали отвечать и вызвали эту тревогу)

В дальнейшем из мнемосхемы для квитирования аварии оборудования достаточно в элемент тревоги записать значение "6": 2 - сбросить бит "Внимание" у элемента тревоги и 4-сбросить признак "Внимание" у всех контролируемых элементов.

9.2.6 Архивирование

Каждый элемент модели, наследующийся от базового, имеет возможность архивирования. Элемент может быть заархивирован как событие аналог или счетчик. Существует несколько типов архивирования, которые делятся на периодические и непериодические. Если для элемента указан периодический тип архивирования, то для него должны быть указаны один или несколько периодов архивирования. Для одного элемента модели можно задать несколько типов

архивирования. Для архивируемого элемента модели создается столько **архивных элементов**, сколько для него указано типов архивирования.

При просмотре в хранилище архивные элементы имеют названия, которые формируются из названия элемента модели и **суффикса**, соответствующего типу данного архивного элемента.

Существуют следующие типы архивирования:

- событий :
 - **Д** - по изменению, суффикс ""
- аналогов :
 - **АС** – текущее значение, суффикс "_тек"
 - **АМ** – среднее значение, суффикс "_сред"
 - **АМн** – максимальное значение, суффикс "_мин"
 - **АМх** – минимальное значение, суффикс "_макс"
 - **А** – по изменению, суффикс ""
- счетчиков:
 - **СМ** – текущее значение, суффикс "_тек"
 - **СР** – разность, суффикс "_разн"
 - **С** – по изменению, суффикс ""

Периодические архивы делятся на те, которые записывают время начала периода архивирования и те, которые записывают время конца периода архивирования.

Принципы работы типов архивирования:

- События по изменению - В этом случае архивирование производится при изменении значения элемента модели или переходе его в недостоверное состояние.
- Аналог среднее. - Архив периодический. Во время работы накапливается среднее значение элемента и периодически архивируется. При записи в архив задается время начала периода архивирования.
- Аналог и счетчик текущее. - Архив периодический. Архивируется первое достоверное значение элемента модели за период архивирования как аналог или счетчик. Записывается время конца периода архивирования.
- Аналог минимальное(максимальное). - Архив периодический. Во время работы модели определяется минимальное (максимальное) значение элемента за период и архивируется. Записывается время начала периода архивирования.
- Аналог и счетчик по изменению. - Архив неперіодический. Для такого типа архивирования задается *такт архивирования и загрузка*. Такт архивирования - это время, через которое будет работать архивный элемент. Если оно не указано (равно нулю), то архивный элемент будет работать каждый такт модели. Такт архивирования задается в секундах. Архивный элемент хранит предыдущее значение элемента модели. Каждый такт архивирования, архивный элемент берет новое значение из элемента модели и архивирует его в том случае, если отклонение нового от старого значения больше загрузки.
- Счетчик разность. - Архив периодический. Архивный элемент сохраняет в себе разность Delta и предыдущее значение элемента модели. Каждый такт Delta увеличивается на разность текущего значения элемента модели и предыдущего (на предыдущем такте). Периодически Delta архивируется. Если архивирование работает для типа wdCIn, то Delta увеличивается на wdCIn::Delta и предыдущее значение не используется. Записывается время начала периода архивирования.

При архивировании неперіодических архивов (События по изменению, Аналог и счетчик по изменению) если архивируется элемент Windecont, то есть один из элементов, начинающийся на wd, то если его значение не определено, то в качестве его значения берется код ошибки Windecont, то есть архивируется код ошибки Windecont.

Для работы архивирования в программе Windecont для модели должно быть указано имя хранилища. Перед стартом архивируемой модели хранилище с указанным именем должно быть заранее создано с помощью «Менеджера хранилища».

Если при запуске архивируемой модели не удастся установить соединение с хранилищем или обнаруживаются другие ошибки, связанные с архивированием, то выполнение модели останавливается и появляется сообщение об этом в окне отладки модели Windecont. Текст сообщения определяется архивным сервером.

Название архивного элемента состоит из **полного названия** элемента модели и суффикса. Для объяснения понятия полного названия элемента модели введем такие понятия: пусть элемент А содержится в элементе В, элемент В содержится в элементе С, элемент С находится в D. Название элемента модели может содержать, а может не содержать специальные символы : '.', '%', '\

- Если название элемента А не содержит специальных символов, то полное_название_А = полное_название_В\название_А. Символ '\' выполняет функцию разделителя. Если элементы В и С также не содержат специальных символов, а D - самый верхний элемент модели, то полное_название_А = название_С\название_В\название_А.

- Если название A имеет вид '\xxx', где xxx - некоторая строка, то полное_название_A = xxx, то есть '\' в начале названия означает, что название абсолютное и не зависит от названий вышестоящих элементов.
- Если название A имеет вид '..xxx', то полное_название_A = полное_название_Vxxx, то есть символ '..' означает пропуск уровня (элемента A) при определении полного названия элемента A.
- Если название A имеет вид '..\..xxx', то полное_название_A = полное_название_Sxxx, то есть пропускаются два уровня (элементы A и B), так как указаны два символа '..', через разделитель '\'. Таким образом, в определении полного названия элемента A будет пропущено столько уровней, сколько встретится в начале названия элемента A символов '..', разделенных символом '\'.
- Если нужно указать в названии символ '\' или '..', то нужно воспользоваться символом '%', который снимает специальный смысл следующего за ним символа. То есть название вида '..\%.xxx' даст полное_название_A = полное_название_S..xxx. Символ % в полном названии отсутствует.

Некоторые вопросы по программированию архивирования см в разделе [Программирование\Архивирование](#).

9.2.7 Элементы OPC

В библиотеке существует группа типов, соответствующих базовым типам (iInt, iBool и т.д) значения которых связаны со значениями элементов одного или нескольких OPC-серверов. Называются эти типы, так же как и базовые с добавлением префикса 'iOpc': iOpcBool, iOpcInt и т.д. Для указания OPC-сервера существует тип **tOpcSrv**. Каждый элемент iOpcXXX принадлежит какому-то одному tOpcSrv. Связь элементов iOpcXXX с tOpcSrv осуществляется следующим образом: элемент iOpcXXX имеет поле **OpcSrvID(int)** которое обозначает идентификатор сервера. Тип tOpcSrv имеет также поле **OpcSrvID(int)**, так что если эти поля OpcSrvID совпадают, то iOpcXXX принадлежит tOpcSrv. Для того чтобы элемент iOpcXXX принадлежал серверу tOpcSrv необходимо также чтобы элемент iOpcXXX находился "ниже" элемента tOpcSrv в структуре модели.

Поля типа tOpcSrv:

- **OpcSrvID(int)** - идентификатор сервера. По умолчанию 0.
- **Machine(AnsiString)** - Имя компьютера в сети, на котором работает OPC-сервер. Если имя компьютера Machine пустое, то сервер находится на локальном компьютере.
- **ProgID(AnsiString)** - Имя OPC-сервера. Имена серверов: для OPC-сервера Windecnt "OPC.Dep.1", для OPC-сервера модели "Dep.Model.1".
- **ReConnectTime(float)** - Через какое время восстанавливать соединение. Указывается в секундах и обозначает время, между разрывом соединения с сервером и повторным с ним соединением. По умолчанию 5.
- **Delimiter(AnsiString)** - Символ-разделитель. Строка, участвующая в образовании имени элемента сервера. Как она участвует, будет пояснено позднее. Для OPC-сервера Windecnt разделитель ':', для OPC-сервера модели '\'.
- **State(iBool)** - состояние работы Opc сервера. Равно true в том случае, когда соединение с OPC сервером установлено и все элементы модели, OpcSrvID которых равно OpcSrvID типа tOpcSrv, успешно установили соединение с элементами OPC сервера. В остальных случаях State равно false.

Каждый элемент iOpcXXX наследуется от элемента iXXX и имеет следующие поля :

- **OpcSrvID(int)** - идентификатор сервера. По умолчанию 0.
- **OpcName(AnsiString)** - Opc-имя элемента. Поле OpcName участвует в получении полного OPC имени элемента (в терминологии стандарта OPC ItemID). Полное OPC имя получается примерно также как и полное название базового типа. Чтобы понять образование полного Opc-имени нужно ввести понятие иерархии элементов типа iOpcXXX. Считается что элемент A типа iOpcXXX содержится в элементе B типа iOpcYYY, если элемент A содержится в элементе B или содержится в элементе, который содержится в элементе B или и т.д.
 - Если OpcName имеет вид '\xxx', то полное имя будет 'xxx'.
 - Если OpcName имеет вид 'xxx', то находится первый в иерархии модели элемент iOpcXXX, который содержит данный элемент, берется его полное Opc имя и складывается с '\' + OpcName.
 - Если OpcName имеет вид '..xxx', берется первый в иерархии элемент iOpcXXX содержащий данный, берется его полное OPC-имя и складывается с 'xxx'. То есть две точки в начале OpcName означают пропуск уровня в иерархии iOpcXXX.
 - Если OpcName имеет вид '..\..xxx', то пропускается два уровня в иерархии.
- **RWType(int)** - Тип чтения-записи элемента. Возможные значения типа чтения-записи:
 - 1 – элемент используется на чтение, то есть его значение равно значению OPC-элемента, на который он указывает.
 - 2 – элемент используется на запись, то есть при записи в элемент происходит запись в OPC-элемент, на который он указывает.
 - 3 – элемент используется и на чтение и на запись. В этом случае при записи в элемент значение записывается и в OPC-элемент. Если же значение OPC-элемента меняется, то меняется значение и самого элемента iOpcXXX.

Если элемент имеет тип RWType равным 2 или 3, то при записи в элемент и отсутствии связи с OPC-сервером, записанное значение все-таки будет записано в сервер при восстановлении соединения. Если во время отсутствия связи с OPC-сервером было несколько записей в элемент, то при восстановлении соединения в OPC-сервер будет записано последнее значение.

Возможность для работы с OPC-сервером есть также и для элементов Windecont. Для этого в типы wdNoA, wdNoD, wdNoC (wdNoX) добавлено поле **OpсSrvID(int)** - номер OPC-сервера.

- Если OpсSrvID равно нулю (значение по умолчанию), то wdNoX означает номер элемента базы в локальном контроллере Windecont.
- Если OpсSrvID не равно нулю, то wdNoX означает номер элемента базы в удаленном контроллере Windecont, который определяется элементом tOpсSrv, у которого OpсSrvID равно OpсSrvID данного wdNoX.

Все типы для работы с базой Windecont, которые имеют поле типа wdNoA или wdNoD или wdNoC могут работать с удаленным контроллером Windecont. Например, тип wdAOut имеет поле No(wdNoA). Если No.OpсSrvID не равно нулю, wdAOut будет работать так же как при OpсSrvID равным нулю, только результирующее значение будет передано не в локальный Windecont, а в удаленный, определяемый типом tOpсSrv с OpсSrvID равным wdAOut.No.OpсSrvID.

При настройке двух элементов wdNoX с разными OpсSrvID, эти элементы будут рассчитываться независимо. То есть, если, например, для wdNoX был задан относительный номер и OpсSrvID не равный нулю, то при подсчете значения wdNoX будут рассматриваться элементы wdNoBase только такие, у которых соответствующее поле wdNoX имеет тот же OpсSrvID.

9.2.8 Оперативный журнал

Оперативный журнал позволяет архивировать последовательность событий и квити́ровать их. Запись журнала содержит следующие поля:

int	ID	уникальный в течение суток идентификатор не равный 0
TDateTime	DateTime	зимнее время возникновения события
String	Action	действие, характеризующее событие, например "Работа", "Останов", "Открытие" и т.д.
String	Object	имя элемента, связанного с событием, например, "КП1\Насос2"
String	Disp	имя диспетчера
WORD	Severity	Важность события от 1(Уведомление) до 1000(Авария)
String	OPCName	полное OPC имя элемента Object, например, "CP1\Pump2"
String	Comment	комментарий
bool	Kvit	квити́ровано ли событие
TDateTime	KvitDateTime	зимнее время квити́рования
String	KvitDisp	Диспетчер, квити́ровавший событие.

Оперативный журнал создается в том же хранилище, в которое происходит архивирование данных модели.

Для работы с оперативным журналом необходимо добавить элемент типа `tOperLog`. Если создать два таких элемента, то при старте модели будет выдана ошибка.

Любой базовый элемент модели содержит поля "**Важность**" и "**Квитировать**". "Важность" - это число от 0 до 1000. "Квитировать" - это флаг (true или false).

Запись в оперативный журнал происходит для элемента при установке в нем тревожного состояния, если его "Важность" больше нуля.

По умолчанию важность равна 0, флаг "Квитировать" не установлен. Чтобы элемент участвовал в формировании записей в оперативном журнале, нужно задать важность больше нуля. Чем больше важность, тем важнее событие. Флаг "Квитировать" указывает на необходимость квитирования события. Если "Квитировать"=true, то при установке у элемента тревожного состояния, помимо записи в оперативный журнал, информация о событии сохраняется с самим элементом-источником события.

Имя диспетчера - это [список текущих пользователей](#).

Более подробно по программированию оперативного журнала см. в разделе [Программирование\Оперативный журнал](#).

Для работы с оперативным журналом создан компонент для среды C++Builder.

Написать про отображение регистрации пользователей в оперативном журнале

9.2.9 Пользователи

Для работы с пользователями в библиотеке модели предусмотрен тип **tUsers**. Элемент `tUsers` содержит (в себе) список возможных пользователей.

Если в модели есть элемент типа `tUsers`, то пока OPC-клиент не регистрируется, он не может писать в элементы модели. Исключение элемента `tUsers`, через который производится регистрация.

Каждый пользователь имеет следующие атрибуты:

- имя,
- пароль,
- права (Администратор или Пользователь) Пользователи делятся на 2 класса: **Пользователи** и **Администраторы**.
Пройдя регистрацию, Администраторы могут менять (добавлять, удалять, менять параметры) список возможных пользователей, а Пользователи могут лишь менять свой пароль. Таким образом, Пользователи отличаются от Администраторов только тем, что первые не могут, а вторые могут менять список пользователей в типе `tUsers`.
- номер категории. По номеру определяется, к какой категории принадлежит пользователь.

Тип `tUsers` имеет поля:

- **AdminPwd**(AnsiString) - пароль Администратора
- **List** - Список возможных пользователей системы. Через "Конструктор OPC-модели" не заполняется.
- **Count**(int) - определяет количество элементов в списке List. По умолчанию равно 10.
- **CtgList** - Список категорий.
- **ClientName**(AnsiString) - имя клиента. При регистрации пользователя в запросе передается строка, которая сравнивается с ClientName. Если они не совпадают, то после регистрации OPC-клиент не имеет доступа на запись в элементы модели.

Чтобы иметь возможность работы нескольких пользователей, в программе "Конструктор OPC модели" нужно добавить в описание модели элемент типа `tUsers` и притом только один. И заполнить поле **AdminPwd** (`AnsiString`). Местоположение элемента этого типа в иерархии и его имя неважно. Если добавить два элемента типа `tUsers`, то при старте модели будет выдана ошибка. При первом старте модели в список возможных пользователей заносится Администратор с именем "Администратор" и паролем, заданным в поле `AdminPwd`. То же самое происходит, если модель стартует при отсутствии сохраненных данных. Поле `AdminPwd` не видно через OPC-интерфейс.

Кроме списка пользователей тип `tUsers` содержит список **категорий** пользователей (тип `tUserCtg`). Каждая категория имеет следующие поля:

- **CtgID**(`iInt`) - номер категории
- **CtgCaption**(`iString`) - название категории.
- **OperLog**(`iBool`) - признак включения пользователя в **список текущих пользователей**.

Пользователь с категорией равной 0 не имеет доступа для записи в модель. Рекомендуется использовать категорию 0 для Наблюдателя.

Список текущих пользователей содержит список имен пользователей, зарегистрированных в данный момент, **номер категории которых не равен нулю** и в категории которых признак `OperLog` установлен в `true`.

При работе с оперативным журналом изменение списка пользователей регистрируется в оперативном журнале.

Регистрация и выход пользователя попадают в журнал с важностью 20.

Добавление, изменение параметров, удаление, изменение порядка в списке пользователей попадают в журнал с важностью 5.

Для работы с пользователями создан компонент для среды C++Builder.

9.2.10 Программирование

Библиотека модели написана на языке программирования Borland C++ и использует библиотеку VCL.

Соглашения по именам

Названия функций и типов библиотеки подчиняются следующим правилам:

- базовые типы начинаются с префикса «i», например, `iInt`, `iBool` и т.д.
- классы общего пользования начинаются с префикса «t», например `tOpcVal`
- глобальные функции и переменные начинаются с префикса «g», например `gLogMessage()`.
- специфические функции и классы имеют специфический префикс, например классы для работы с элементами базы WinDecont имеют префикс «wd».
- интерфейсы начинаются с префикса "i", например `iArItem`.

Обзор

Классы и функции библиотеки делятся на специфические и общего пользования. К общим классам относят

- пары **tPairT** – пары значение + его достоверность + время.
- вариантный тип **tVar**, который может содержать значение фундаментальных типов.
- OPC-значение **tOpcVal**, содержащий значение типа `tVar`, OPC-качество и время.

Элементы модели, видимые через интерфейс OPC, наследуются от класса [tBase](#), который

- содержит значение, качество и время, то есть OPC-значение
- в котором определены параметры узла дерева модели,
- обеспечивает возможность прохода по узлам дерева
- реализует такие возможности элемента модели как
 1. сохранение значения при рестарте модели
 2. архивирование значения элемента
 3. работа с оперативным журналом

На основе типа tBase и пар определены базовые типы, содержащие значения [фундаментальных](#) типов:

iVoid, iWord, iInt, iFloat, iString, iDateTime и т.д. Тип iVoid не содержит значения и используется только как узел дерева модели.

Замечания

Если для типов имеет смысл оператор присваивания, то он, как правило, работает. Например, можно писать так:

```
void f()
{
    vInt v; tOpcVal v2, tVar v3;
    v2 = v; // можно
    v2 = 1.1; //
    v3 = v;
}
```

Чтобы присвоить значение элементу базового типа из его функции (например, tact) пишете так (*this) = v;

Выделенное значение имеет функция **assign**. Если есть некоторый тип tType, который характеризуется параметрами типа tType1 и tType2, то тип tType, как правило, имеет функцию void tType::assign(const tType1& a1, const tType2 a2); Например, пара vInt характеризуется параметрами int (значение) и bool (достоверность), поэтому есть функция void vInt::assign(int aValue, bool aGood); Функция assign в этом случае заменяет на новые значение и достоверность vInt.

При наследовании слово **inherited** означает базовый класс. Это используется при написании функций обработки, например функция tact некоторого типа tType

```
void tType::tact()
{
    inherited::tact(); // обработка по умолчанию.
    if (a){ b = c + d; }
}
```

Порядок работы модели

Модель имеет тактовый принцип работы. Такт - это время, через которое запускается некоторая функция обработки модели. Каждый элемент модели имеет свою функцию обработки, так что функция обработки всей модели пробегает по всем элементам.

1. Выполнение конструктора корневого элемента модели.
2. Восстановление сохраненных данных
3. Пользовательская настройка afterCreate
4. Сброс динамических флагов tBase
5. Такт
 - Если нужно, пересчет и проверка номеров wdNoX
 - Если нужно, установка связей OPC элементов с серверами по OpcSrvID.
 - tact
 - afterTact

- Сохранение измененных сохраняемых элементов.
 - Сброс динамических флагов tBase
6. Вызов beforeDelete.
 7. Выполнение деструктора корневого элемента.

Запись в Windecont происходит во время выполнения стадии afterTact.

9.2.10.1 Архивирование

Элемент модели может быть архивируемым. В архивировании участвуют следующие поля tBase:

```
const AnsiString Caption; // название
DWORD          ArchMask; // маска архивирования
tArPeriods*   Periods; // периоды архивирования для периодических архивов.
tEnumTable*   EnumTable; // таблица расшифровки для событий
float          ArchZagr; // загрузка для аналогов и счетчиков по изменению
float          ArchTact; // период для аналогов и счетчиков по изменению.
```

С помощью названия образуется имя элемента в архиве. Полное архивное имя элемента создается соединением полного названия и суффикса. Как формируется полное название см. [определение полного названия](#).

Маска архивирования. Биты маски указывают на тип архивирования элемента.

Существуют следующие типы архивирования

событий :

Д 0x01 - по изменению, суффикс ""

аналогов :

АС 0x04 – текущее значение, суффикс "_тек"
 АМ 0x02 – среднее значение, суффикс "_сред"
 АМн 0x08 – минимальное значение, суффикс "_мин"
 АМх 0x10 – максимальное значение, суффикс "_макс"
 А 0x80 – по изменению, суффикс ""

счетчиков:

СМ 0x20 – текущее значение, суффикс "_тек"
 СР 0x40 – разность, суффикс "_разн"
 С 0x100 – по изменению, суффикс ""

Элемент архивируется, если маска не равна нулю.

Для каждого типа архивирования создается объект типа **iArItem**. Если тип архивирования периодический, то для него создается столько объектов iArItem, сколько указано периодов в поле Periods. Объект iArItem подготавливает и выполняет архивирование. Как он это делает – зависит от его типа.

События по изменению. Каждый такт работы модели объект iArItem проверяет признак tBase::event() и если он установлен, то выполняет архивирование.

Аналог среднее. Архив периодический. Каждый такт модели iArItem подсчитывает сумму значений tBase и увеличивает на 1 количество измерений. Если вышло время архивирования, то выполняется архивирование суммы, деленной на количество измерений.

Аналог и счетчик текущее. Архив периодический. Когда выходит время архивирования, iArItem архивирует значение tBase.

Аналог минимальное(максимальное). Архив периодический. Объект iArItem сохраняет в себе текущее минимальное (максимальное) значение. Каждый такт модели объект iArItem сравнивает новое значение tBase с сохраненным и если оно меньше(больше), сохраняет новое значение. Когда выходит время архивирования, iArItem архивирует сохраненное значение.

Аналог и счетчик по изменению. Архив периодический, но период берется из поля tBase::ArchPer. Период задается в секундах. Объект iArItem хранит предыдущее значение архивируемого элемента. Когда выходит время архивирования, объект iArItem берет новое значение из tBase и архивирует его в том случае, если отклонение нового от старого значения

больше `tBase::ArchZagr`.

Счетчик разность. Архив периодический. Объект сохраняет в себе разность `Delta` и предыдущее значение `tBase`. Каждый такт `Delta` увеличивается на разность текущего значения `tBase` и предыдущего (на предыдущем такте) значения `tBase`. Когда выходит время архивирования `iArItem` архивирует значение `Delta`. Если архивирование работает для типа `wdCIn`, то `Delta` увеличивается на `wdCIn::Delta` и предыдущее значение не используется.

Чтобы найти объект `iArItem` по его типу и периоду используется функция

```
iArItem* tBase::findArItem(DWORD aTypeID, int aPeriod);
```

типа `tBase`. Тип `iArItem` содержит следующие функции:

- `void addValue(const tOpcVal& v);` // добавить значение в архив
- `void setActive(bool); bool active();` // признак активности

объект `iArItem` архивирует значение в соответствии со своим типом только, если он активен. Такое поведение может использоваться для архивирования, контролируемого пользователем, например:

```
void tType::tact()
```

```
{
    inherited::tact();
    iArItem* ai = findArItem(0x02,60); // находим элемент архивирования (среднее значение аналога раз в минуту)
    ai->setActive(false); // отключаем архивирование по умолчанию
    tOpcVal val = 123; // архивируемое значение
    ai->addValue(val); // архивирование

    ai->setActive(true); // включаем архивирование по умолчанию, если это нужно.
}
```

Таблицы расшифровки

Для работы с таблицами расшифровки существуют следующие типы:

tEtRec - структура, состоящая из двух полей:

- **ID**(int) - номер состояния
- **Str**(String) - строка, соответствующая данному номеру

Переменная типа `tEtRec` находится в `tEnumTable`. При удалении `tEtRec` (вызове оператора `delete`) переменная удаляется из списка в типе `tEnumTable`.

tEnumTable - тип, содержащий таблицу расшифровки. `tEnumTable` предоставляет следующие поля:

- `AnsiString ArName;` - имя таблицы
- `int ArID;` - номер таблицы

и функции:

- `unsigned count();` - возвращает кол-во элементов в таблице
- `tEtRec* getRec (DWORD aIdx);` - возвращает строку по индексу `aIdx`. Индекс принимает значение от 1 до `count()`. Если индекс недопустим, то возвращается 0.
- `tEtRec& addRec (int aID, const AnsiString& aStr);` - добавляет новый элемент в конец таблицы.
- `tEtRec* recByID (int aID);` - возвращает строку по значению поля `ID` типа `tEtRec`. Если элемент не найден то возвращается 0.

Для удаления элемента из таблицы нужно получить на него указатель `tEtRec*` и вызвать над ним оператор `delete`.

Для создания новой таблицы расшифровки нужно

- в функции `afterCreate()` создать тип `tEnumTable`,
- заполнить таблицу элементами с помощью функции `addRec`,
- установить указатель таблицы в поле `EnumTable` типа `tBase` вызовом функции `void setEnumTable(tEnumTable*);`
- в функции `beforeDelete()` удалить таблицу.

Если таблица (переменная типа `tEnumTable`) создана статически, то первый и последний шаги не нужны.

Для редактирования существующей (созданной в программе Конструктор OPC-модели) таблицы нужно:

- Выяснить имя переменной типа `tEnumTable`, соответствующей таблице. Имя переменной задается так: **gEnTableNo**, где No - номер таблицы. Например, для таблицы с номером 1010, имя переменной будет `gEnTable1010`.
- в функции `afterCreate` изменить таблицу, используя функции типа `tEnumTable`.

Периоды архивирования

Для работы с периодами архивирования в библиотеке существует тип **tArPeriods**. Тип содержит список периодов архивирования. Период представляет собой целое число со знаком.

Если для периода задано положительное число, то период архивирования равен этому числу в секундах.

Если задано

- -1 - период архивирования - календарный месяц.
- -2 - период архивирования - квартал
- -3 - период архивирования - год

Тип `tPeriods` имеет следующие функции:

- void **addPer**(int aPer); - добавляет период `aPer` в конец списка.
- void **delPer**(unsigned aIdx); - удаляется период с индексом `aIdx` из списка. `aIdx` может принимать значение от 1 до `count()`
- int **getPer**(unsigned aIdx); - возвращает значение периода по индексу `aIdx`
- void **setPer**(unsigned aIdx, int aPer); - устанавливает период `aPer` по индексу `aIdx`
- unsigned **count**()const; - возвращает количество периодов.

Периоды архивирования, созданные в программе Конструктор OPC-модели, именуются следующим образом: **gPeriodsNo**, где No - номер списка периодов. Изменить список периодов можно только в функции `afterCreate`. Для создания нового списка периодов нужно:

Для создания новой таблицы расшифровки нужно

- в функции `afterCreate()` создать переменную типа `tPeriods`,
- заполнить ее периодами с помощью функции `addPer`,
- установить указатель переменной в поле `ArchPeriods` типа `tBase`.
- в функции `beforeDelete()` удалить переменную.

Если таблица (переменная типа `tPeriods`) создана статически, то первый и последний шаги не нужны.

9.2.10.2 Работа со временем

В библиотеке определены глобальные переменные:

const `TDateTime` **gUtcTime**; текущее UTC-время

const `TDateTime` **gLocTime**; текущее локальное время

const TDateTime **gLocWTime**; текущее локальное зимнее время.

Эти переменные пересчитываются перед выполнением такта. UTC(Universal Coordinated Time) – время по Гринвичу.

Функции перевода времени:

TDateTime **gUtcToLocal**(TDateTime aUTCTime); // UTC-время в локальное время

TDateTime **gUtcToLocalW**(TDateTime aUTCTime); // UTC-время в зимнее время

TDateTime **gCurUtcTime**(); //текущее UTC-время

TDateTime **gLocalToUTC**(TDateTime aDT); // локальное в UTC-время

TDateTime **gLocalWToUTC**(TDateTime aDT); // зимнее в UTC-время

TDateTime **gDecontTimeToDateTime**(DWORD d);// время DECONT в TDateTime

DWORD **gDateTimeToDecontTime**(TDateTime dt); // TDateTime в время DECONT

Типы библиотеки модели, являющиеся [парами](#), содержат UTC-время. При создании переменной типа пары в ней устанавливается по умолчанию текущее UTC-время.

9.2.10.3 Пары

Парами называют семейство типов, каждый из которых содержит значение, его достоверность и время. Для каждого базового типа существует своя пара, например, для int – vInt, для AnsiString – vString и т.д. Каждая пара имеет следующие функции:

- value_t **value**();// возвращает значение пары, где value_t – тип значение(int, bool, AnsiString и т.д)
- bool **good**(),**bad**(); // проверка на достоверность значения
- void **assign**(value_t val, bool aGood=true);// установить новое значение и достоверность. Если не использовать второй параметр, то значение будет достоверно
- void **makeBad**();// сделать значение недостоверным.
- TDateTime **utcTime**(); // получить UTC-время
- void **setUtcTime**(); // установить UTC-время
- void **assign**(value_t aValue, bool aGood, TDateTime aDT);// установить новые значение, достоверность и время. Третий параметр можно опустить, тогда будет установлено текущее время.

Пары можно использовать так:

```
void f()
{
    vInt a(5,true); // создается пара a с достоверным значением 5
    vInt b(5); // то же самое.
    a = 3; // после присвоения в a находится достоверное значение 3;
    a.makeBad(); // a становится недостоверным
}
```

Пара vBool отличается от остальных тем, что ее можно использовать на чтение как переменную типа bool, которая равна vBool::value() && vBool::good(); Например:

```
void f()
{
    vBool a(true,true);
    if (a){
        // сюда перейдет управление, так как значение a достоверно и равно true.
    }
}
```

Над парами можно выполнять арифметические операции. Эти операции введены для того, чтобы не анализировать достоверность операндов, а только лишь, если это необходимо, анализировать достоверность результата, что сокращает запись алгоритмов.

- Численные операции: **+, -, *, /**. Результат – пара, тип значения которой – «максимальный» из типов значений операндов. Правила определения «максимального» такие же, как в языке C++. При выполнении операции значение результата равно операции над значениями операндов, а качество равно «логическому И» над качествами операндов, то есть результат будет достоверным, только если оба операнда достоверны.

Например:

```
vInt a; vFloat b;
```

Выражение a+b. Тип выражения будет vFloat так как тип float «больше» чем int. Выражение эквивалентно функции:

```
vFloat operator+( vInt a, vFloat b)
{
    return vFloat(
        a.value()+b.value(), // значение результата
        b.good() && a.good() // достоверность результата
        max(a.utctime(),b.utctime()) // время результата выбирается как максимальное из двух времен.
    );
}
```

- Логические операции: **>, <, >=, <=, ==, !=, &&, ||**. Результат – пара, тип значения которой bool. При выполнении операции значение результата равно операции над значениями операндов; для всех операций кроме || («логическое ИЛИ») качество равно «логическому И» над качествами операндов, то есть результат будет достоверным, только если оба операнда достоверны. Для операции || если среди операндов есть достоверный и со значением true, то значение пары-результата равно true и достоверно; в противном случае (если нет достоверного операнда со значением true), то значение пары операнда неопределено и недостоверно. Исключением является случай, когда оба операнда достоверны и равны нулю: в этом случае результат будет достоверным, а значение равно нулю. Например:

```
vInt a; vFloat b;
```

Выражение a>b. Тип выражения будет vBool. Выражение эквивалентно функции :

```
vBool operator>( vInt a, vFloat b)
{
    return vBool(a.value()>b.value(),b.good() && a.good(),max(a.utctime(),b.utctime()));
}
```

Выражение a||b. Тип выражения будет vBool. Выражение эквивалентно функции :

```
vBool operator||( vInt a, vFloat b)
{
    bool res = (a.value() && a.good()) || (b.value() && b.good());
    return vBool res,a.good() || b.good(),max(a.utctime(),b.utctime());
}
```

[Базовые типы](#) (iInt, iBool и т.д.) так же являются парами.

Тип tVar

Тип tVar наследуется от стандартного типа VARIANT и имеет переопределенные арифметические операции. Тип VARIANT имеет поле vt типа VARTYPE, которое определяет тип содержащегося в VARIANT значения. Таким образом, в VARIANT могут содержаться значения разных типов: целые числа, числа с плавающей точкой, строки. VARIANT так же может не содержать никакого значения, то есть быть пустым. При этом vt=0. При создании переменной типа tVar без инициализации она будет пустой. Использовать tVar можно так:

```
void f()
```

```
{
    tVar v; // в v ничего нет.
    v = "строка" // в v строка «строка»
    v = 5; // в v целое значение 5
    tVar v2 = 5.5, r, emp;
    r = v + v2; // в r будет результат сложение – число 10.5
    if (r>v){
        // управление перейдет сюда, так как r больше v
    }
}
```

```

    if (r > emp) {
        // управление сюда не перейдет, так как emp пустое и нельзя сказать, что r
        // больше emp.
    }
    r = v + 3.3; // результат – число 8.3
}

```

Тип **tOpcVal**

Тип **tOpcVal** является парой со временем, то есть входит в семейство типов **tPairT**. Тип значения этой пары – **tVar**. **tOpcVal** содержит OPC-качество, по которому определяется достоверность значения. Так как **tOpcVal** – это пара, то над **tOpcVal** можно производить арифметические операции. Приведем функции **tOpcVal**:

- **tVar&** **value()**; // значение
- **WORD** **quality()**; // OPC-качество
- **TDateTime** **utcTime()**; // время
- **void** **assign(const tVar& aValue, WORD aQ, TDateTime aDT);** // Установка нового значения, качества и времени. Если опустить параметр **aDT**, то установится текущее время, если опустить параметр **aQ**, то установится достоверное значение.

9.2.10.4 Базовые элементы

Тип **tBase**

Все элементы модели, видимые через интерфейс OPC, наследуются от типа **tBase**. Наследуемый от **tBase** тип может иметь значение (**iInt**) или не иметь его (**iVoid**). Работает оператор присваивания для типа [tOpcVal](#):

```
void f(tBase& b) { tOpcVal v = b; b = v; }
```

Элементы модели располагаются в виде дерева. Для прохода по узлам дерева в типе **tBase** предусмотрены поля:

- **tBase* First**; // первый элемент, содержащийся в данном.
- **tBase* Next**; // следующий элемент на одном уровне с данным.
- **tBase* Owner**; // элемент, в котором содержится данный.

Для прохода по подэлементам можно пользоваться такой конструкцией:

```

void f(iInt& v)
{
    for(tBase* i = v.First; i=i->Next){
        // переменная i – указатель на элемент, содержащийся в v.
    }
}

```

Тип **tBase** содержит имя и название:

- **const AnsiString Name**; // Имя.
 - **const AnsiString Caption**; // Название.
 - **void setName(const AnsiString& aName);** // Установить новое имя.
 - **void setCaption(const AnsiString& aCapt);** // Установить новое название.
- Если в строке **aCapt** встречается пара CR LF ("\r\n"), то она заменяется на пробел.
- **AnsiString fullName();** // Получить полное имя элемента.
 - **AnsiString fullCaption();** // Получить полное название элемента.
 - **tBase* findByName(const AnsiString& aStr);** // Поиск подэлемента по его имени.

Полное имя получается сложением полного имени **Owner**, разделителя «\» и имени **Name** самого элемента. Если **Owner** – самый верхний элемент модели, то полное имя равно просто **Name** самого элемента.

Установка имени и названия не должны вызываться в тактовой функции обработки.

С помощью функции **findByName** можно искать элементы не только на данном уровне, но и глубже, используя в строке символ-разделитель уровня '\'. То есть если параметр функции имеет вид 'aaa\bbb', то на данном уровне будет найден элемент с именем 'aaa', и если он найден, то в нем будет найден элемент 'bbb'. Если функция не находит элемента с заданным именем, то она возвращает 0.

Тип **tBase** содержит следующие виртуальные функции обработки

- **virtual void afterCreate();** // вызывается после создания. В этой функции выполняется настройка элемента и запрос

ресурсов.

- virtual void **tact()**; // тактовая функция
- virtual void **afterTact()**; // послетактовая функция
- virtual void **beforeDelete()**; // вызывается перед удалением элемента. Освобождаются ресурсы, выделенные в afterCreate.

При наследовании от tBase (или от другого класса, который наследуется от tBase) эти функции могут быть переопределены, выполняя обработку специфичную для данного типа. По умолчанию, эти функции проходят по всем подэлементам, то есть при вызове функции tact самого верхнего элемента, будут вызваны функции tact у всех элементов дерева, если пользовательская обработка не отключила выполнение функции tact типа tBase.

Пусть Т – самый верхний элемент модели. Порядок вызова функций обработки такой: сначала создается элемент Т (вызывается его конструктор), затем вызывается Т::afterCreate(), затем запускается таймер, каждый тик которого вызывается сначала Т::tact(), затем Т::afterTact(). При окончании работы модели сначала останавливается таймер, затем вызывается Т::beforeDelete() и удаляется элемент Т.

Функции tact и afterTact по смыслу отличаются только порядком выполнения, то есть тактовую обработку можно написать в любой из них, если не важен взаимный порядок выполнения обработки данного элемента и остальных. Пример переопределения функции обработки:

```
class A : public tBase {};
void A::tact()
{
    // До вызова тактовых функции подэлементов
    ...
    inherited::tact(); // Вызов тактовых функций подэлементов
    // После вызова тактовых функций подэлементов
    ...
}
```

Для обозначения родителя в наследовании для типов модели принято имя типа inherited. В данном случае inherited= tBase. Подэлементы – это элементы нижних уровней иерархии. Метод tBase::tact() работает так, что, проходя по всем подэлементам tBase (а значит и А), вызывает их метод tact.

Дополнительные возможности типа tBase.

Элементы, наследуемые от tBase, могут быть замаскированы. Замаскированный элемент не виден через интерфейс OPC и над ним не выполняются функции обработки, так как функции обработки tBase определены так, что функции замаскированных подэлементов не вызываются.

- void **setMasked**(bool v); // Установить замаскирован элемент или нет.
- bool **masked**(); // Возвращает true если замаскирован данный элемент, или он содержится в замаскированном.
- setMasked нельзя вызывать в тактовых функциях обработки, то есть, замаскирован или нет элемент- должно быть определено на стадии функции afterCreate.

Элементы tBase могут быть сохраняемыми.

- void **setStored**(); Функция делает элемент сохраняемым и должна быть вызвана в конструкторе элемента.

Тип tBase имеет функции

- void **logMessage**(const AnsiString& aStr);
- void **fatalError**(const AnsiString& aStr);

которые аналогичны глобальным функциям gLogMessage и gFatalError, только к строке aStr будет добавлено полное имя элемента tBase.

Базовые типы.

На основе типа tBase в библиотеке введены базовые типы, содержащие значение фундаментальных типов. Базовый тип является парой соответствующего фундаментального типа со временем и реализует возможности типа tBase.

Базовыми типами являются: iVoid, iBool, iInt, iWord, iDWord, iFloat, iDouble, iString, iDateTime. Эти типы являются базовыми классами большинства всех элементов модели, доступных по интерфейсу OPC. Тип iVoid не содержит значения.

Каждый базовый тип содержит следующие флаги:

- bool **alarmed**(); void **setAlarmed**(bool); Признак «тревожного» состояния элемента. Участвует в образовании OPC-качества элемента.
- bool **written**(); void **setWritten**(bool);

Признак записи некоторого значения в элемент после предыдущего такта работы. Например:

```
struct A : iInt
{
    iInt v;
    void tact()
    {
        inherited::tact();
        v = 2;
        // После оператора v=2 v.written()==true.
    }
};
```

Флаг взводится как при записи в элемент в методах обработки, так и при записи через OPC-интерфейс. Сбрасывается в конце такта после afterTact.

- bool **vChanged()**; void **setVchanged**(bool); Признак того, что изменилось значение элемента.
- bool **qChanged()**; void **setQchanged**(bool); Признак изменения OPC-качества элемента. Взводится при изменении alarmed и good.
- bool **gChanged()**; void **setGchanged**(bool); Признак того, что изменилась достоверность значения (результат метода good()). Это происходит или при записи нового значения в элемент, когда он имеет недостоверное значение, или после makeBad. Участвует в образовании OPC-качества элемента.
- bool **event()**; void **setEvent**(bool); Взводится при gChanged, vChanged и при взводе alarmed.

Флаги written, vChanged, qChanged, gChanged, event являются динамическими, то есть сбрасываются в false в конце каждого такта после afterTact. Затем могут быть взведены в true в результате OPC-функций записи и чтения или работы методов tact и afterTact.

9.2.10.5 Ссылки, списки, массивы

Ссылка tLink.

[Ссылка](#) на некоторый элемент работает как указатель (в терминах языка C++) на этот элемент.

Ссылка имеет функцию setLink для установки элемента, на который она указывает. Оператор '->' возвращает указатель на элемент, а оператор '*' возвращает ссылку (в терминах C++) на элемент, на который ссылка ссылается. Например:

```
struct A : iInt
{
    iInt b;
    tLink<iInt> c;
    A():iInt()
    {
        // Инициализация ссылки.
        c.setLink(&b); // ссылка c указывает на элемент b.
    }
    void tact()
    {
        inherited::tact();
        c->assign(2); // эквивалентно b.assign(2);
        // или
        *c = 2; // то же что и b = 2;
    }
};
```

Ссылка имеет следующие функции (T - параметр ссылки):

- void **setLink**(T* a) устанавливает указатель на элемент
- T* **link**() возвращает указатель на элемент
- T* **operator->**() обеспечивает оператор -> над ссылкой и возвращает указатель на элемент
- T& **operator*** () обеспечивает оператор * над ссылкой и возвращает C-ссылку на элемент

Массив

Массив имеет функцию **unsigned count()** возвращающую количество элементов массива и переопределенный оператор **[]** возвращающий i-тый элемент массива.

Пример работы с массивом:

```
void f()
{
    tArray<iInt,5> arr5; // tArray<iInt,5> - тип массива из 5 элементов типа iInt
    for(int i=1;i<=arr5.count();i+)// функция count() возвращает количество элементов в массиве.
    {
        arr5[i] = i;// присваивание i-тому элементу массива.
        iInt& val = arr5[i]; // получение ссылки на i-тый элемент.
    }
}
```

Список

Список имеет следующие поля и функции:

- tBase* **NodeFrom**; //Узел, начиная с которого будет происходить заполнение списка. Если узел не указан (NULL) то заполнение начнется с Owner.
- unsigned **count()**; //Количество элементов в списке.
- tLink<T>& **operator[]**(unsigned i) //Возвращает элемент с индексом i.
- void **add**(T& i); //Добавляет элемент в конец списка.
- unsigned **indexOf**(T& item); //Возвращает индекс элемента. Если результат – 0, то такого элемента в списке нет.
- void **clear**(); //Очистка списка.
- void **afterCreate**(); //В методе afterCreate происходит заполнение списка на основе NodeFrom и T.

В случае, когда необходим список значений элементов, тип которых заранее неизвестен, используется список элементов типа tBase, а для указания типа заполнения используется дополнительная функция. Пример:

```
class A : iInt
{
    tList<tBase> List;
    void afterCreate()
    {
        inherited::afterCreate();
        List.NodeFrom = this;
        List.fillType<iFloat>();// Указание на то, что заполнять список элементами типа iFloat начиная с элемента A.
    }
};
```

9.2.10.6 Элементы Windecont

Для работы с элементами баз существуют типы **wdDiscret**, **wdAnalog**, **wdCounter**.

Пусть T – тип элемента базы (T=WORD для дискретов, T=float для аналогов, T=DWORD для счетчиков). Каждый из этих типов имеет следующие поля:

- WORD **No**; // Номер в базе
- WORD **State**; // Состояние элемента
- T **Value**; // Значение

Следующие функции работают над полями No, State и Value;

- bool **valid**(); // Возвращает true, если Windecont работает и элемент с номером No существует.

- bool **good**(); // Значение достоверно
- bool **bad**(); // Значение недостоверно
- bool **isDyn**(); // Установлен бит динамики.
- void **assign**(T aVal, bool aGood=true, bool aDyn=false) // Установка значения, достоверности и признака динамики.
- T **value**(); // Возвращает значение Value;

Следующие функции работают с базой Windecont.

- void **read**(); // Чтение эл-та
- void **write**()const; // Запись значения Value в базу.
- void **write**(T aVal); // Запись значение aVal в Value, а затем в базу
- void **get**(); // Получение значения элемента.
- void **set**(); // Установка значения Value в базу.
- void **set**(T aVal); // Установка значение aVal в Value, а затем в базу.
- Для счетчиков существуют дополнительные функции:
- void **setInc**(DWORD delta); // Инкремент
- void **dec**(DWORD newVal, DWORD oldVal); // Декремент.

Каждый тип wdAnalog, wdDiscret, wdCounter имеет статическую переменную:

static DWORD **Count**; содержащую количество элементов в базе.

Помимо перечисленных в разделе [Элементы Windecont](#) полей, типы wdDIn, wdDOut, wdAOut имеют поле **D**(bool) - признак динамики. Тип wdDIn при чтении из базы Windecont устанавливает поле D в true, если в дискрете установлен бит динамики. Поле D держится равным true на протяжении одного такта работы модели, затем сбрасывается в false. Для типов wdDOut и wdAOut если установлено поле D в true, то в элемент базы записывается значение с установленным битом динамики, после чего поле D сбрасывается в false.

Элементы для работы с базами Windecont делятся на входные (wdAIn, wdDIn, wdCIn) и выходные (wdAOut, wdDOut, wdDOutImp, wdCOut). Входные осуществляют свою обработку (чтение из базы Windecont и т.д.) в функции tact(), а выходные (запись в базу Windecont) в функции afterTact().

Типы [wdNoD](#), [wdNoA](#), [wdNoC \(wdNoX\)](#) для установки параметров имеют функцию

```
void assign( WORD aSubNo, tBase* aFrom, bool aAbs, int aOpcSrvID);
```

которая изменяет параметры wdNoX, причем на следующем такте работы модели перед тактовой функцией главного элемента номера wdNoX будут пересчитаны по новым параметрам, и элемент будет связан с сервером aOpcSrvID.

Присваивание элементу wdNoX равносильно вызову функции assign с параметрами aSubNo = присваиваемое значение, aFrom = NULL, aAbs = true, aOpcSrvID = текущий OpcSrvID элемента wdNoX. То есть :

```
void f(){
    wdNoA FNo; FNo = 5; // эквивалентно FNo.assign( 5, NULL, true, FNo.OpcSrvID );
}
```

Например, тип wdNoD имеет следующее объявление:

```
struct wdNoD : iWord
{
    tBase* From; // Элемент, относительно которого будет происходить пересчет номера
    bool Abs; // Признак того, что для подсчета номера элемента используется абсолютная нумерация.
    WORD SubNo; // Под-номер. Если нумерация абсолютная, то значение wdNoD станет равным SubNo, иначе будет взят
    используемый относительный номер и к нему добавлен SubNo.
    int OpcSrvID; // номер OPC-сервера Windecont, если значение элемента берется из удаленного OPC-сервера
    Windecont.
};
```

Приведенные поля могут использоваться только для чтения. Для записи используется функция void **assign**(WORD aSubNo, tBase* aFrom, bool aAbs, int aOpcSrvID);

Например, для изменения поля SubNo, нужно вызвать функцию assign таким образом:

```
void f(){
```

```

wdNoD v;
int NewSubNo; // Новое значение номера SubNo.
v.assign(NewSubNo,v.From,v.Abs,v.OPCSrvID);
}

```

9.2.10.7 Тревоги

Тип `alBase` имеет следующие поля:

- `List (tList<tBase>)` - список контролируемых элементов. Это поле имеет тип - список ссылок.
- **LastAllList (vList<bool>)** - Список тревожных состояний на предыдущем такте.

и функцию

- virtual bool **checkAlarm**(unsigned i)=0; для определения тревожного состояние i-того в списке List элемента. Переопределяется в наследуемых от `alBase` классах.

```
typedef alBase alUser;
```

Пользовательский класс тревоги. Для определения своего типа тревоги нужно пронаследоваться от `alUser` и переопределить функцию `checkAlarm`. Например, для типа `alValue` `checkAlarm` выглядит следующим образом:

```

bool alValue::checkAlarm(unsigned index)
{
    // List[index] – ссылка на tBase
    // cur – текущее значение контролируемого элемента с индексом i
    tVar cur = List[index];
    // prev – предыдущее значение
    tVar &prev = PrevValList[index];
    if (cur.bad()) return false;
    if (prev.bad()){
        prev = cur;
        return false;
    }
    // сравниваем предыдущее и текущее значения
    bool result = cur!=prev;
    // обновляем элемент списка предыдущих значений
    PrevValList[index]=cur;
    // возвращаем то, в тревожном ли состоянии находится i-ый элемент, то есть изменилось ли его значение.
    return result;
}

```

При программном заполнении списка `List` контролируемых элементов следует предусмотреть следующее:

- после его заполнения нужно также заполнить списки, зависящие от списка `List`. Например, в типе `alBase` - это список `LastAllList`. В типе `alValue` к нему добавляется список `PrevValList`. По крайней мере, количества элементов в списках должны совпадать.
- Нужно правильно проинициализировать списки. Список `LastAllList` инициализируется так:
`for(unsigned i=1;i<=LastAllList.count();i++) LastAllList[i] = checkAlarm(i);`

9.2.10.8 Элементы OPC

Тип `tOPCSrv` имеет следующие дополнительные поля:

```
struct tOPCSrv
{
    ...
    int LastDisconnectHR; // код ошибки, указывающий на причину разрыва связи с сервером в последний раз.
};
```

Каждый элемент `iOpcXXX` имеет следующие поля:

```
{
    ...
    int Item->State; // состояние элемента. Возможные значения: 0 - Начальное состояние, 1 - Соединение установлено, 2 - Была попытка соединения, но неуспешно, 3 - Есть данные(хотя бы однажды значение элемента было получено от орс сервера)
    int Item->LastHR; // код ошибки последней операции над элементом.
};
```

9.2.10.9 Оперативный журнал

Рассмотрим частичные объявления основных классов, участвующих в работе оперативного журнала.

```
struct tOLPars // параметры события.
{
    ...
    iBase*   Base;
    TDateTime DateTime;
    WideString Action;
    WideString Object;
    WideString Disp;
    WORD     Severity;
    WideString OPCName;
    WideString Comment;
    bool     KvitNeed;

    tOLPars(tBase* v=0, const WideString& Act=L"");
};
```

В конструкторе `tOLPars`

- `DateTime` устанавливается в текущее время;
- `Disp` устанавливается в текущего диспетчера, используя глобальную функцию `curDisp()`;
- `Action` равно параметру `Act`;
- `Base` равно переменной `v`.

После этого если `Base` не равно 0, то

```
Object = Base->fullCaption(), то есть Object равно полному архивному имени элемента;
OPCName = Base->fullName(), то есть OPCName равно полному OPC-имени элемента;
Severity = Base->AISeverity;
KvitNeed = Base->alkvitNeed;
```

Если Action - пустая строка, то Action равно значению элемента Base, преобразованное в строку.

Преобразование значения элемента tBase в строку происходит следующим образом:

```
{
  tBase* Base = ...;           // исходный элемент
  vString str = *Base;        // Получаем пару типа vString
  if (str.good()){            // Если преобразование успешно то
    AnsiString res = str.value(); // получаем искомую строку.
  }
  // Если элемент Base имеет целочисленный тип и таблицу расшифровки, то преобразование в строку будет с учетом этой
  // таблицы.
}

struct tOperLog : iVoid
{
public :
  iString Action; // Элемент для квитирования событий OPC-клиентом

  DWORD add(const iOLPars& Pars); // добавить запись
  bool kvit(DWORD ID, DWORD day); // Квитировать событие по ID и DateTime
  bool kvit(iBase& v);           // Квитировать все события элемента v
} extern *gOperLog;             // Глобальная переменная для работы с iOperLog.
```

Глобальная переменная **gOperLog** указывает на единственный экземпляр tOperLog. Если в модели нет элемента tOperLog, то gOperLog=0.

Используя поле Action OPC, клиент может квитировать события. Запись в Action строки "kvit1,ID,Day", где ID - идентификатор события, Day - день события (целая часть TDateTime), эквивалентна вызову метода Kvit(DWORD ID, DWORD day).

Пусть, iInt LogItem - элемент модели, через запись в который происходит запись в журнал.

Чтобы программно записать событие в журнал и потом квитировать его нужно сделать примерно следующее:

```
if (LogItem.written())
{
  tOLPars pars;
  // Заполняем параметры pars
  pars.Action = ...;
  pars.Severity = ...;
  pars.Disp = gCurDisp(LogItem); // пользователь, который записал в LogItem.
  ...
  DWORD id=0; // идентификатор записи.
  // Добавляем запись в журнал
  if (gOperLog)
    id = gOperLog->add(pars);
  // Если после этого id==0, то записи в журнал не произошло.
  ...
  // Квитируем
  if (id!=0 && gOperLog)
    gOperLog->kvit(id,pars.DateTime); //
}
}
```

Чтобы наш код работал и в случае отсутствия оперативного журнала, мы проверяем gOperLog на равенство 0.

```
struct tAllItem // информация о неквитированном событии
{
  DWORD ID;
  TDateTime DateTime;
```

```
String Action;
};

struct tBase : ...
{
    ...
    WORD alSeverity(); // Авария или предупреждение 1-1000
    bool alKvitNeed(); // Нужна ли квитанция

    AnsiString fullName()const; // соответствует OPCName
    AnsiString fullCaption()const; // соответствует Object
    ...
};
```

tBase содержит список неквитированных событий tAllItem.

OPC-клиент через OPC-свойство может прочитать список неквитированных событий tAllItem.

При рестарте модели список tAllItem сохраняется.

Далее считается, что tBase::alSeverity()=0

Перевод элемента в тревожное состояние осуществляется методом setAlarmed(bool v) с параметром true.

Квитация элемента происходит при вызове setAlarmed с параметром false.

Вот примерный код метода setAlarmed:

```
void iBase::setAlarmed(bool v)
{
    if (v){ // Генерация события
        // Заполняем параметры
        iOLPars pars(this);
        // Action равно значению элемента, преобразованного в строку.

        if (gOperLog) gOperLog->add(pars);
    } else { // Квитирование всех событий данного элемента
        if (gOperLog) gOperLog->kvit(*this);
    }
}
```

Рассмотрим, что происходит при вызове функций add и kvit типа tOperLog.

- **DWORD add(const tOLPars& Pars);** При вызове этой функции в оперативный журнал добавляется запись с параметрами из переменной Pars. Функция возвращает уникальный в течение суток идентификатор ID записи, который затем может использоваться при квитировании. Если Pars.Base не равно нулю и Pars.Base->alKvitNeed(), то полученный ID, DateTime и Action в типе tAllItem сохраняет с элементом tBase. Таким образом, в tBase заносится информация о неквитированном событии.
- **bool kvit(DWORD ID,DWORD day);** По ID и day(день возникновения события) находится элемент tBase, содержащий неквитированное событие tAllItem с такими параметрами, и это событие tAllItem удаляется из iBase. Признак тревоги alarmed() у найденного tBase сбрасывается. В оперативном журнале это событие помечается как квитированное. Функция возвращает true, если событие с такими ID и day найдено и успешно квитировано.
- **bool kvit(tBase& v);** Функция отличается от предыдущей тем, что она удаляет и квитирует в оперативном журнале все содержащиеся в tBase неквитированные события tAllItem.

9.2.10.10 Пользователи

Для работы со списком пользователей через OPC-интерфейс, то есть для программ работающих под ОС Windows, нужно использовать элемент tUsers.

Элемент типа [tUsers](#) доступен на запись и используется для выполнения команд:

- добавить пользователя,
- удалить пользователя,
- изменить параметры такого-то пользователя,
- зарегистрировать пользователя и т.д.

Команда кодируется в виде списка строк, разделенных запятыми. Если строка в кавычках не содержит пробелов, запятых или кавычек, то кавычки можно опустить.

- *logon*, "имя пользователя", "пароль", "ClientName"- регистрация. При успешной регистрации если параметр ClientName не совпадает с полем tUsers.ClientName, то пользователь не имеет доступа на запись в элементы модели.
- *logoff* - завершение работы пользователя
- *addUser*, "имя пользователя", "пароль", "доступ", "категория" - добавление пользователя, доступ имеет значения: "0" - обычный пользователь, "1" - Администратор.
- *delUser*, "имя пользователя" - удаление пользователя
- *chPwd*, "имя пользователя", "новый пароль" - изменение пароля пользователя.
- *chCtg*, "имя пользователя", "категория" - изменение категории пользователя.
- *chName*, "имя пользователя", "новое имя" - изменения имени пользователя.
- *chAccess*, "имя пользователя", "новый доступ" - изменение доступа.

Команды addUser, delUser - доступны только администратору.

Команды chPwd, chCtg, chName, chAccess доступны администратору или пользователю, если "имя пользователя" совпадает с именем текущего пользователя.

9.2.10.11 Глобальные переменные и функции

- void **gFatalError**(const String&)
 - останов работы модели с выдачей сообщения в окно программы Windecont
- void **gLogMessage**(const String&)
 - вывод сообщения в окно программы WinDecont
- void **gLoadVals**(tBase* aFrom, bool aSubItems=true);
 - Чтение сохраненных данных для элемента aFrom и содержащихся в нем подэлементов при установленном флаге aSubItems.
- String **gCurDisp**(); "текущий диспетчер".
 - Функция возвращает строку, содержащую информацию о текущем диспетчере. Если вызов gCurDisp происходит в такте модели, то в этой строке находится [список текущих пользователей](#). Если вызов gCurDisp происходит во время записи OPC-клиентом некоторого значения в элемент модели (например, при квитировании тревоги), то строка gCurDisp содержит полное имя зарегистрированного через данный OPC-клиент пользователя.
- String **gCurDisp**(tBase& aBase);
 - Выводит имя пользователя, который последним записал значение в элемент aBase через OPC-интерфейс. При отсутствии такого пользователя возвращает результат функции gCurDisp();
- String **gModelDir**();
 - Возвращает каталог, в котором находится файл Model.dll
- void **gUseKvitUserBegin**(tBase& aBase);
 - После вызова этой функции в оперативный журнал при квитировании будет использоваться пользователь, который последним записал значение в элемент aBase.
- void **gUseKvitUserEnd**(tBase& aBase);
 - Отменяет использование пользователя при квитировании в оперативном журнале функцией **gUseKvitUserBegin**.
- bool **dsctTime**; При установленной переменной для элементов типа wdIn независимо от типа Туре происходит чтение дискрета со временем.
- bool **cntTact**; При установленной переменной в случае если такт модели выполняется меньше чем за timeTact % времени заданного такта и если была попытка чтения дискрета со временем с признаком того что есть еще дискреты со временем, то вслед за выполненным тактом выполняется еще один.

- `int timeTact`; Переменная устанавливает границу в процентах выполнения повторного такта как указано для переменной `cntTact`.

9.3 Программа 'Конструктор OPC-модели'

9.3.1 Введение, определения

ПО "SyTrack-TOOL" OPCModelBuilder (программа "Конструктор OPC-модели") является средой для создания OPC-модели. Оно позволяет описывать объект автоматизации как древовидную структуру элементов, которые доступны через OPC-интерфейс (элементы модели). Кроме описания структуры объекта, с помощью программы "Конструктор OPC-модели" можно также создать алгоритмы управления объектом и алгоритмы архивирования данных.

Программа имеет набор базовых типов и функций, которые можно использовать при построении модели. Кроме этого, "Конструктор OPC-модели" позволяет создавать свои типы и наборы функций для них. Тип, который является образом объекта автоматизации, называется **главный тип**.

Описания типов хранятся в библиотеках. Одна библиотека может содержать несколько описаний типов. Каждая библиотека хранится в отдельном файле с расширением ".dtl". Для каждого типа, описанного в библиотеке можно задать набор функций. Для этого к библиотеке необходимо добавить [файл функций](#). При добавлении файла функций к библиотеке <библиотека.dtl>, в той же папке будут созданы два файла <библиотекаF.cpp> и <библиотекаF.h>. Библиотека, в которой описан главный тип, называется **главной библиотекой**.

Кроме описаний типов, в библиотеке хранятся таблицы справочников. Библиотека может вообще не содержать описания типов, а содержать только таблицы справочников. Справочники - наборы таблиц, которые используются при редактировании некоторых свойств элемента OPC-модели. В настоящее время существуют два типа справочников, таблицы которых хранятся в библиотеках: "[Кодировка дискретов](#)" и "[Периоды архивирования](#)".

Существует еще один справочник "[События оперативного журнала](#)". Этот справочник состоит из единственной таблицы, которая содержится в описании проекта. Описание проекта хранится в файле с расширением ".dep". Каждый проект содержит одну или несколько библиотек. Помимо перечня библиотек, в описании проекта указывается такт работы модели, какой из многочисленных типов является образом объекта автоматизации (главный тип) и многое другое (см. [установки проекта](#)).

9.3.2 Справочники

Справочники - наборы таблиц, которые используются при редактировании некоторых свойств элемента модели. Таблицы справочников "[Кодировка дискретов](#)" и "[Периоды архивирования](#)" хранятся в библиотеках. Единственная таблица справочника "[События оперативного журнала](#)" хранится в описании проекта.

Например, чтобы указать, как будут отображаться различные состояния дискретного сигнала надо сослаться на таблицу кодировки, где все это описано.



Рис. 1

9.3.2.1 Периоды архивирования

В модели есть возможность архивировать значения элементов. Для одного и того же элемента, часто бывает необходимо задать несколько периодов архивирования.

Справочник "Периоды архивирования" - набор таблиц, каждая из которых содержит один или несколько периодов архивирования.

Справочник, показанный на рисунке 1, содержит всего одну таблицу "Периоды архивов". Эта таблица содержит 2 периода: 1 час и 1 сутки. Таблица хранится в библиотеке LibType.

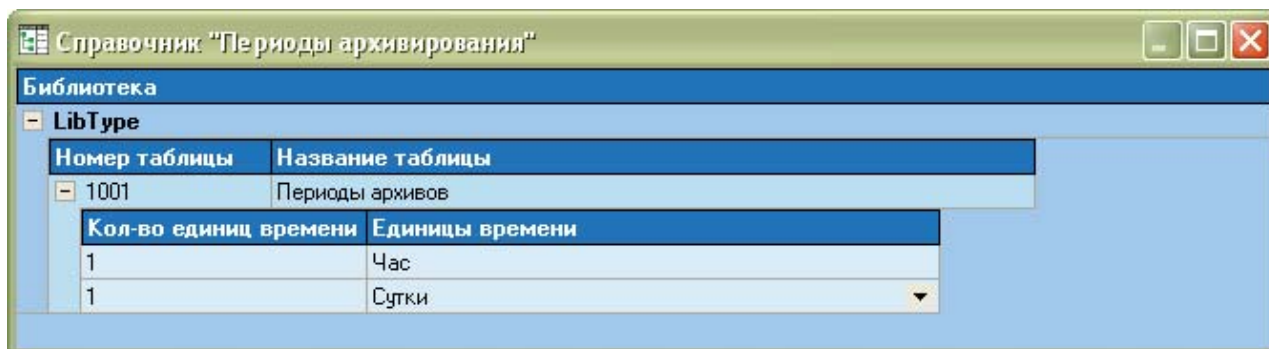


Рис. 1

Таблица "Периоды архивирования" должна отвечать следующим требованиям:

- имя таблицы должно быть уникально;
- имя таблицы должно иметь длину от 1 до 100 символов;
- номер таблицы должен быть уникален;
- номер таблицы должен быть больше 1001;
- на кол-во единиц времени накладываются ограничения:
 - Событие: кол-во единиц должно быть 0;
 - Год, квартал, месяц, сутки: кол-во единиц должно быть 1;
 - Час, минута, секунда: кол-во единиц не должно превышать сутки и должно укладываться в сутках целое число раз.

На рисунке 2 показан тип "ТТр", в котором есть ссылки на таблицу "Периоды архивов". По умолчанию, значения его элементов "cl_Tnv", "cl_T1" ... "cl_T5" будут архивироваться в часовом и суточном архивах.

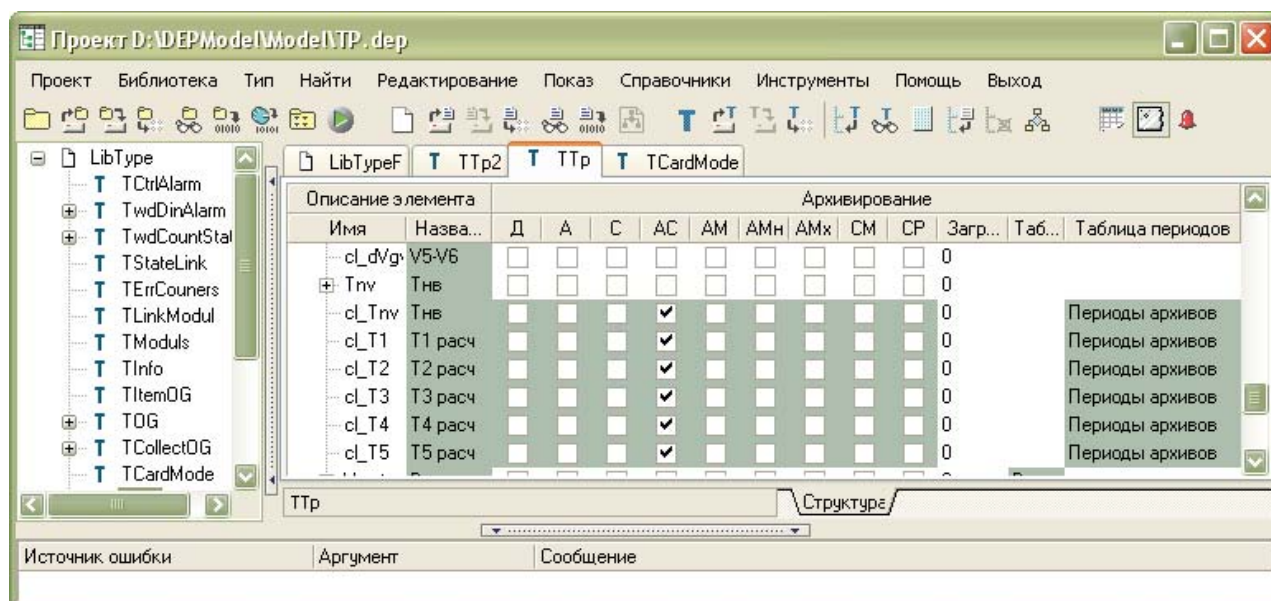


Рис. 2

9.3.2.2 Кодировка дискретов

При просмотре значения дискретного сигнала, удобнее видеть не цифровой код (0, 1, 25), а название соответствующего события ("Остановлен", "Включен", "Блокирован").

Справочник "Кодировка дискретов" - набор таблиц, которые содержат значения дискретов и соответствующие им названия события. В программе predeterminedены несколько таблиц. Они относятся к группе системных таблиц и их нельзя редактировать. В справочнике, показанном на рисунке 1, определены 2 пользовательские таблицы "Ошибки РК" и "Ошибки счета". Обе таблицы хранятся в библиотеке LibType.

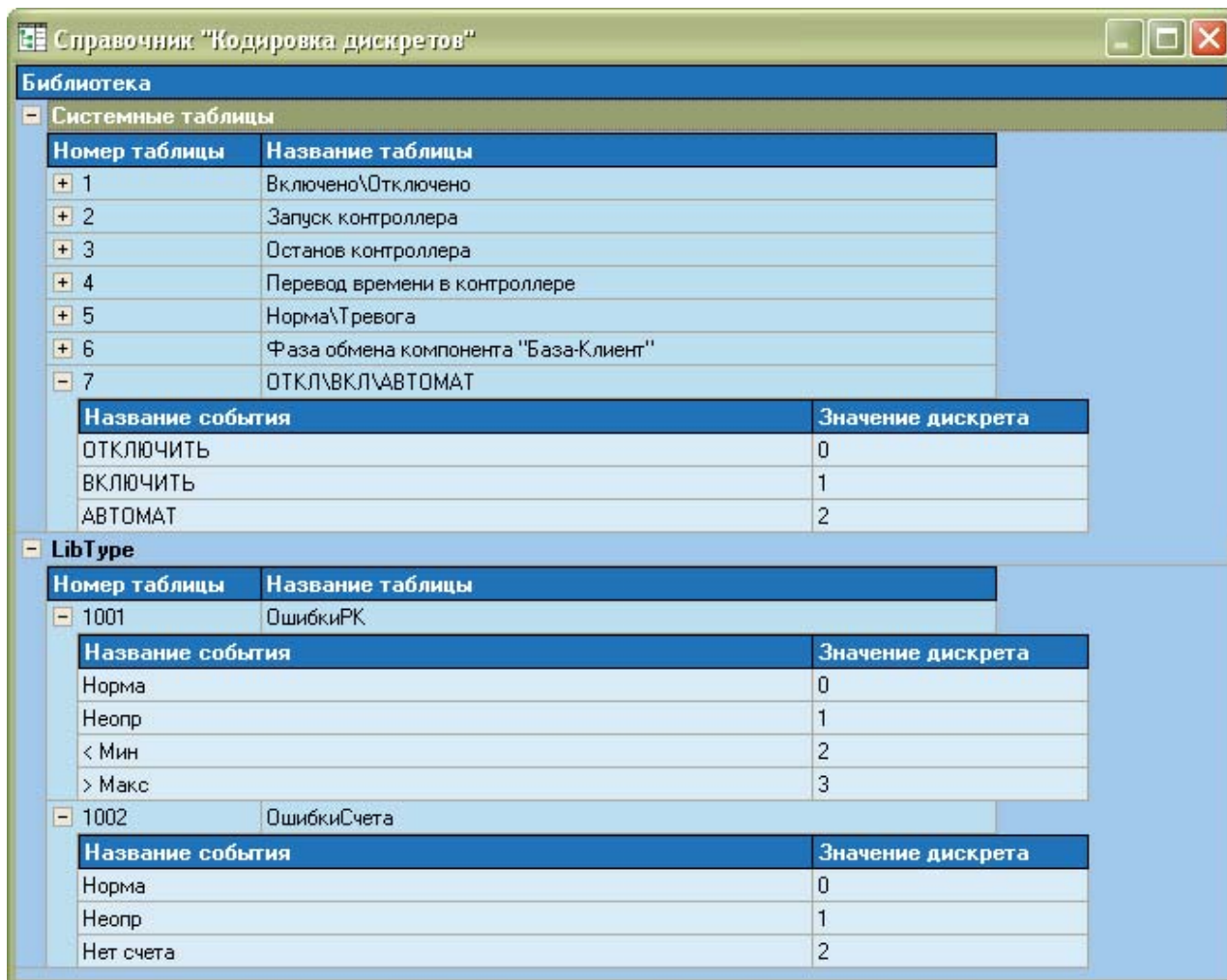


Рис. 1

Таблица "Кодировка дискретов" должна отвечать следующим требованиям:

- имя таблицы должно быть уникально;
- имя таблицы должно иметь длину от 1 до 100 символов;
- номер таблицы должен быть уникален;
- номер таблицы должен быть больше 1001, таблицы с меньшими номерами являются системными и не могут быть изменены;
- название состояния должно иметь длину от 1 до 100 символов;
- код состояния должен быть уникален в рамках одной таблицы.

На рисунке 2 показан тип "TCardMode", в котором есть ссылки на таблицу "Ошибки РК". По умолчанию, значение его элемента "CodeErr" равное 0 будет отображаться как "Норма", значение 1 как "Неопр" и т.д.

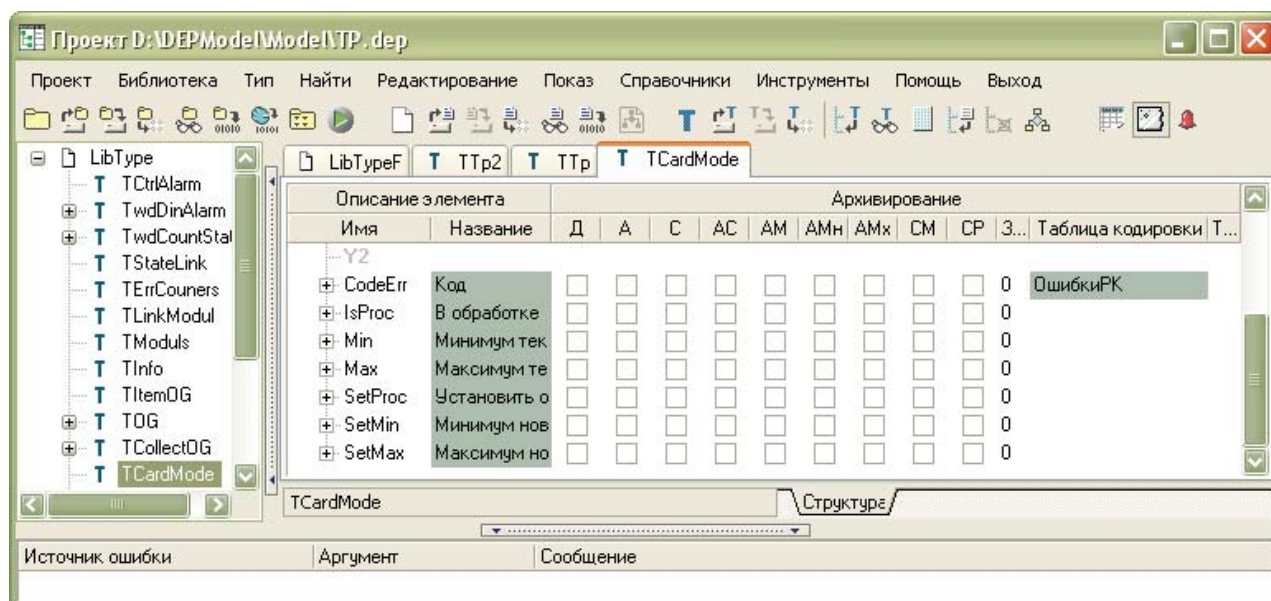


Рис. 2

9.3.2.3 События оперативного журнала

Модель может вести оперативный журнал, куда попадают те или иные события. При просмотре оперативного журнала, удобнее видеть не цифровой код важности, а его символьное обозначение. Справочник "События оперативного журнала" содержит единственную таблицу соответствий между кодом и символьным обозначением важности. Справочник хранится в файле проекта. Название важностей с кодом 5 и 20 предопределены.


Название важности	Важность
Регистрация\Выход пользователя	5
Изменение параметров пользователя	20

Рис. 1

9.3.3 Проект

Каждый проект содержит одну или несколько библиотек. Помимо перечня библиотек, в описании проекта указывается такт работы модели, какой из многочисленных типов является образом объекта автоматизации (главный тип) и многое другое (см. [установки проекта](#)). Проект хранится в файле с расширением ".dep".

9.3.3.1 Новый проект

Чтобы создать новый проект, надо или нажать на кнопку "Новый проект" , или выбрать в главном меню пункт "Проект\Новый проект".

В результате будут созданы проект, библиотека, файл функций и главный тип.

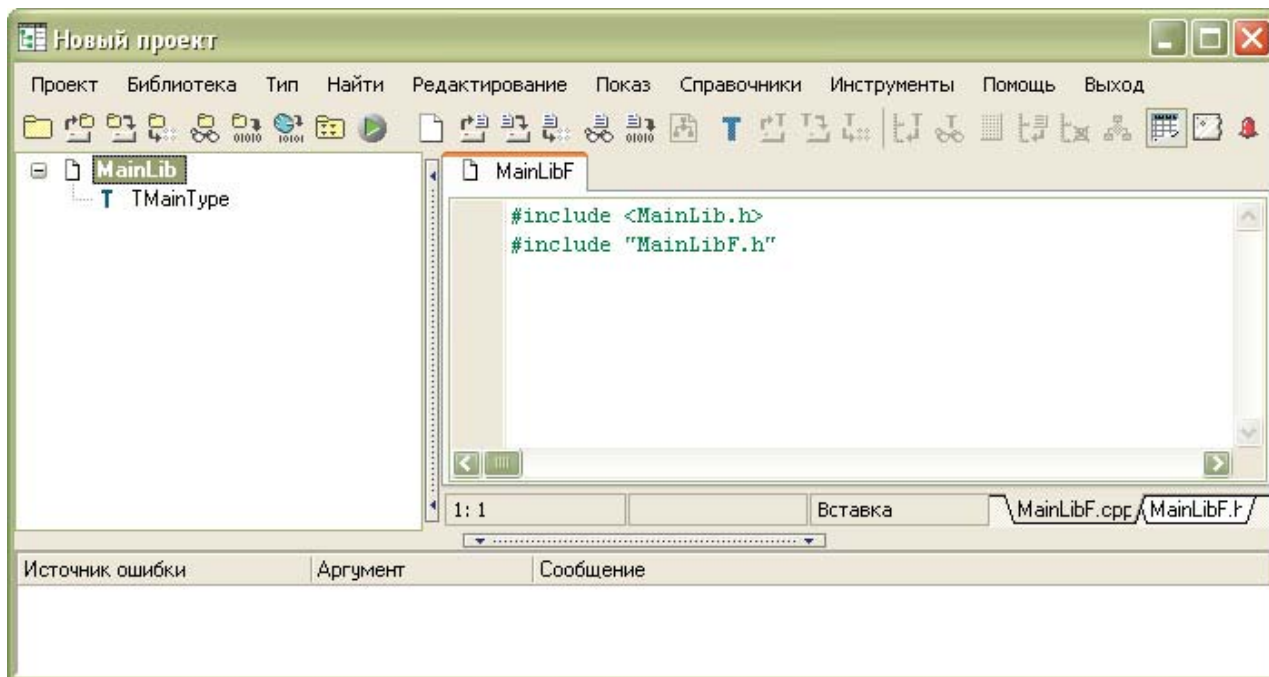



Рис. 1

9.3.3.2 Открыть проект

Чтобы открыть проект, надо или нажать на кнопку "Открыть проект" , или выбрать в главном меню пункт "Проект\Открыть проект", после чего указать имя файла проекта.

Есть более быстрый способ. "Конструктор модели" запоминает пять последних проектов, с которыми он работал. Чтобы открыть один из них, надо выбрать имя проекта в пункте "Проект" главного меню.

При открытии проекта, открывается и его главная библиотека.

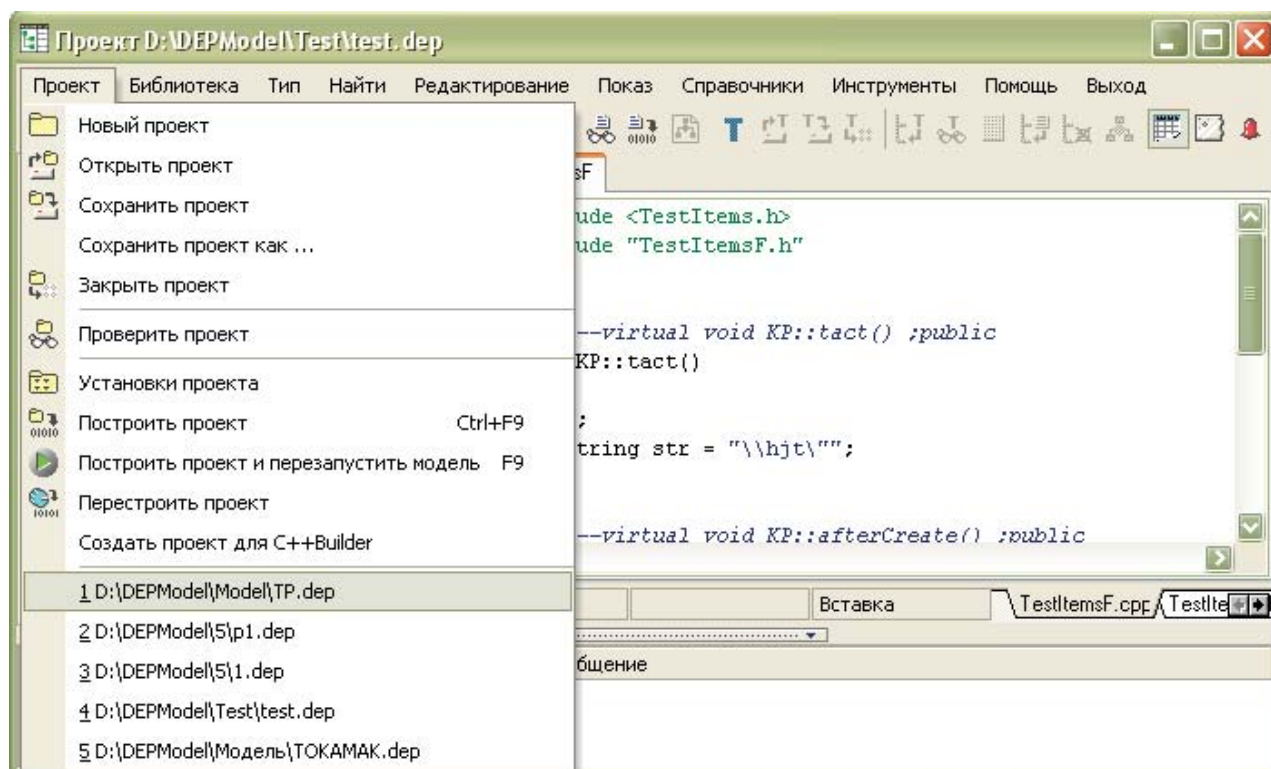



Рис. 1

9.3.3.3 Сохранить проект

Чтобы сохранить проект, надо или нажать на кнопку "Сохранить проект"  или выбрать в главном меню пункт "Проект\Сохранить проект" или "Проект\Сохранить проект как ...".

При сохранении проекта:

- сохраняются все открытые библиотеки;
- сохраняются [установки проекта](#).

При сохранении нового проекта или, если выбрано "Сохранить проект как ..." необходимо дать ему имя и указать папку, где он будет размещен.

9.3.3.4 Установки проекта

Чтобы попасть в установки проекта надо или нажать на кнопку  или выбрать в главном меню пункт "Проект\Установки проекта".

Экран "Установки проекта" содержит три закладки: "Общее", "Библиотеки" и "Построение".

Закладка "Общее" (Рис 1) содержит описание следующих параметров:

- Главный тип - имя типа, который образом объекта автоматизации, другими словами, самой моделью. В качестве главного

типа можно выбрать любой тип, описанный в главной библиотеке.

- Архивное хранилище - псевдоним хранилища, где будут архивироваться значения элементов модели.
- Такт (мс) - период вызова тактовых функций элементов модели.

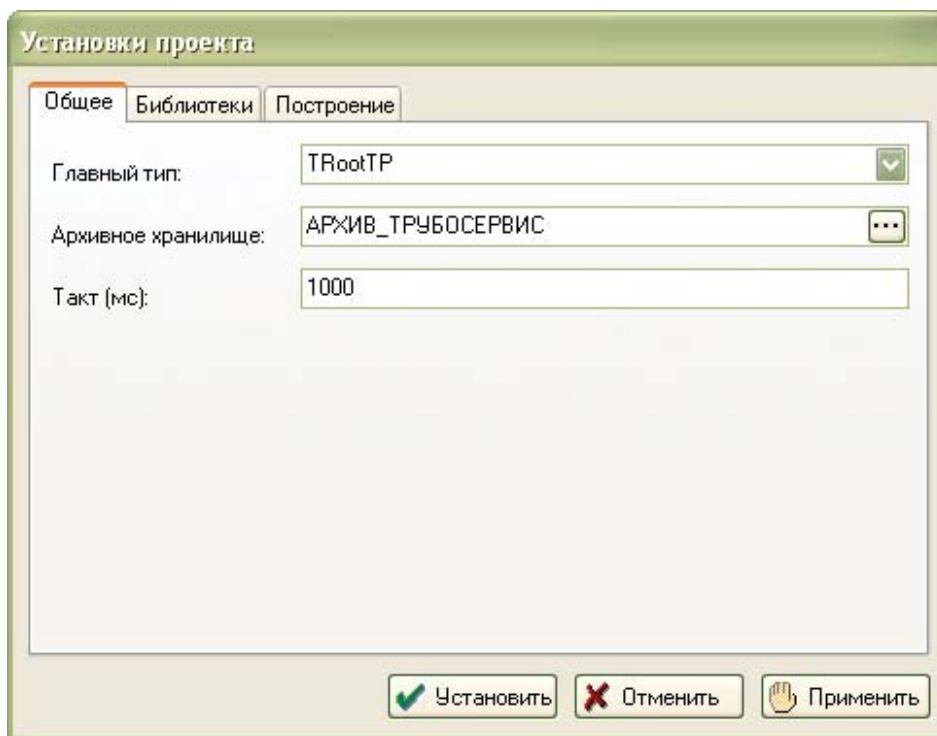


Рис. 1

Закладка "Библиотеки" (Рис 2) содержит список всех библиотек, которые используются в данном проекте. Здесь же указывается, какая из них является главной.



Рис. 2

Закладка "Построение" (Рис 3) содержит дополнительную информацию, необходимую для построения модели.

- файлы .c - исходные файлы с расширением ".c" или ".cpp", которые будут участвовать в сборке модели;
- файлы .obj - объектные файлы, которые будут участвовать в сборке модели;
- файлы .lib - библиотеки, которые будут участвовать в сборке модели;
- папки INCLUDE - папки, содержащие заголовочные файлы, необходимые при сборке модели;
- папки LIB - папки, содержащие файлы библиотек, необходимые при сборке модели;
- папка вывода - папка, где будет собираться файл модели model.dll.

При указании файлов и папок можно указывать как полные пути, так и пути относительно данного проекта. При указании путей можно также использовать переменные окружения.

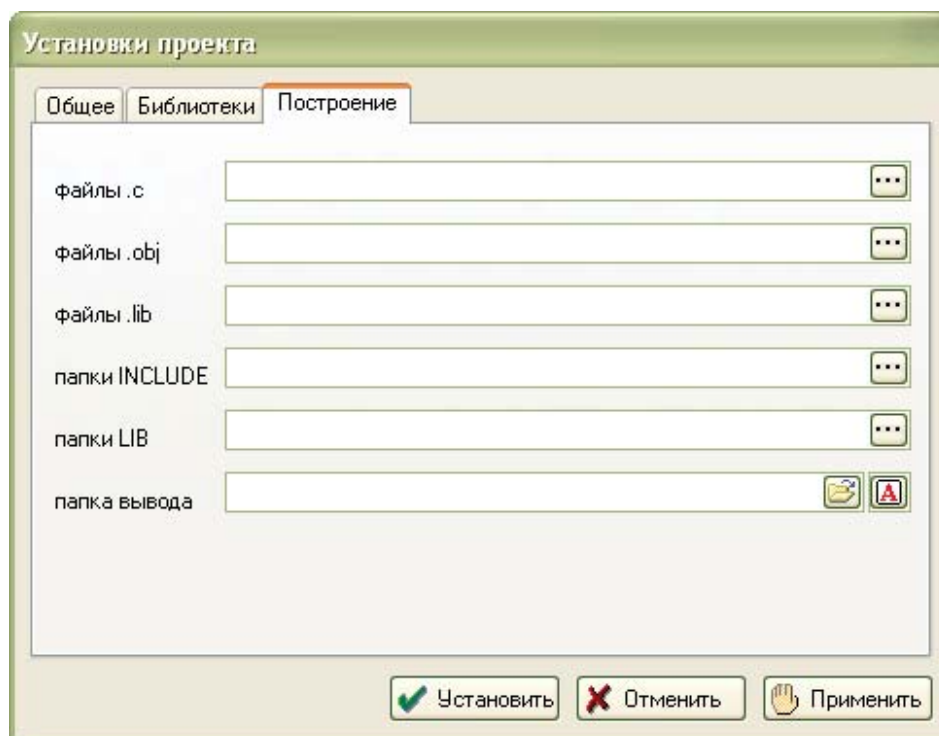



Рис. 3

9.3.3.5 Проверить проект

Чтобы проверить проект, надо или нажать на кнопку "Проверить проект"  или выбрать в главном меню пункт "Проект\Проверить проект".




При проверке проекта:

- проверяется правильность установок проекта;
- проверяются все библиотеки, необходимые для построения модели.

9.3.3.6 Построить проект

Результатом построения проекта является файл model.dll

Существует несколько способов построения проекта:

-  "Проект\Построить проект";
-  "Проект\Построить проект и перезапустить модель";
-  "Проект\Перестроить проект" - перестроить проект.

При построении/перестроении проекта:

- выполняется проверка проекта. Если во время проверки будут обнаружены ошибки, построение не производится;
- для всех библиотек проекта:
 - выполняется проверка библиотеки. Если во время проверки будут обнаружены ошибки, построение не производится;

- создается файл <библиотека.h>, в котором все описанные в библиотеке типы реализуются в виде описания классов. В описание класса также помещается декларация всех функций, описанных для соответствующего типа в файле функций;
 - создается файл <библиотека.c>, в котором описаны конструкторы для всех классов библиотеки;
 - файлы <библиотека.c> и <библиотекаF.c> компилируются с помощью C++Builder в объектные файлы <библиотека.obj> и <библиотекаF.obj>.
- создаются и компилируются файлы main.c и main.h, именно в них реализуются установки проекта;
 - все необходимые объектные и библиотечные файлы собираются с помощью C++Builder в файл model.dll. Если в [установках проекта](#) не указана папка вывода, файл model.dll будет находиться в папке RESULT.

Если выбрана процедура "Построить проект и перезапустить модель", то после успешного создания файла model.dll, модель будет перезапущена.

Отличие процедур "построить" и "перестроить" заключается в том, что при построении, новые файлы <библиотека.h> и <библиотека.c> создаются только в том случае, если дата создания существующих файлов младше даты изменения файла <библиотека.dtl>. При "перестроении" файлы <библиотека.h> и <библиотека.c> всегда создаются заново.

9.3.3.7 Создать проект для C++Builder

Создать проект для C++Builder можно через пункт меню "Проект\Создать проект для C++Builder".

При этом:


- выполняется проверка проекта. Если во время проверки будут обнаружены ошибки, создание проекта не производится;
- для всех библиотек проекта:
 - выполняется проверка библиотеки. Если во время проверки будут обнаружены ошибки, создание проекта не производится;
 - создается файл <библиотека.h>, в котором все описанные в библиотеке типы реализуются в виде описания классов. В описание класса также помещается декларация всех функций, описанных для соответствующего типа в файле функций;
 - создается файл <библиотека.c>, в котором описаны конструкторы для всех классов библиотеки.
- создаются файлы main.c и main.h, именно в них реализуются установки проекта;
- создается файл model.bpr.

После этого, файл model.bpr можно открыть в C++Builder и именно там продолжить редактирование и сборку модели.

9.3.4 Библиотека

В библиотеках хранятся описания типов и таблицы справочников "[Кодировка дискретов](#)" и "[Периоды архивирования](#)". Одна библиотека может содержать несколько описаний типов и несколько таблиц справочников. Каждая библиотека хранится в отдельном файле с расширением ".dtl". Для каждого типа, описанного в библиотеке можно задать набор функций. Описания функций и сами функции хранятся в файлах <библиотекаF.cpp> и <библиотекаF.h>. Библиотека, в которой описан главный тип, называется **главной библиотекой**.

9.3.4.1 Новая библиотека

Чтобы создать новую библиотеку, надо или нажать на кнопку "Новая библиотека"  или выбрать в главном меню пункт "Библиотека\Новая библиотека". Будет создана пустая библиотека с именем <NewLib_xxx> и файлы функций <NewLib_xxxF.cpp> и <NewLib_xxxF.h> (рис. 1). Вновь созданная библиотека автоматически добавляется в проект.

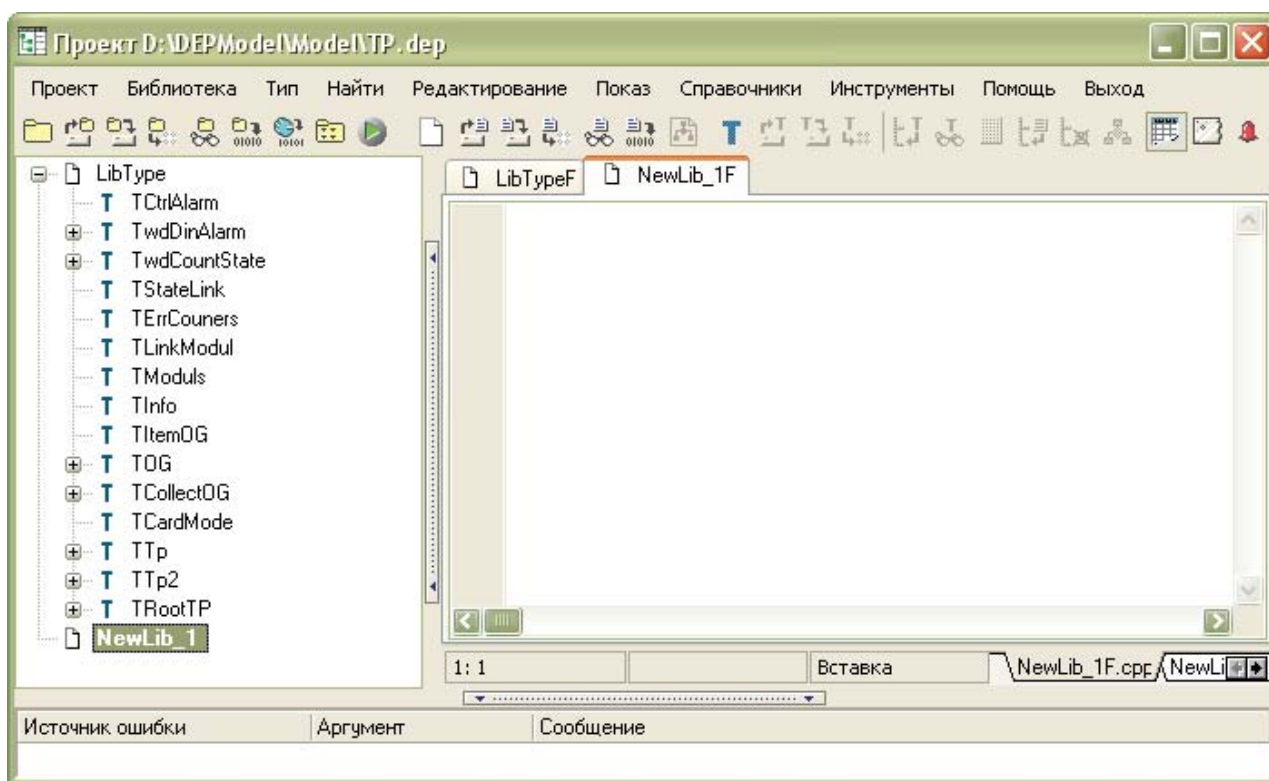



Рис. 1

9.3.4.2 Открыть библиотеку

Главная библиотека проекта открывается при открытии проекта. Чтобы открыть другую библиотеку проекта, надо или нажать на кнопку "Открыть библиотеку"  или выбрать в главном меню пункт "Библиотека\Открыть библиотеку".

Открыть можно только ту библиотеку, которая входит в состав проекта (см. [установки проекта](#)).

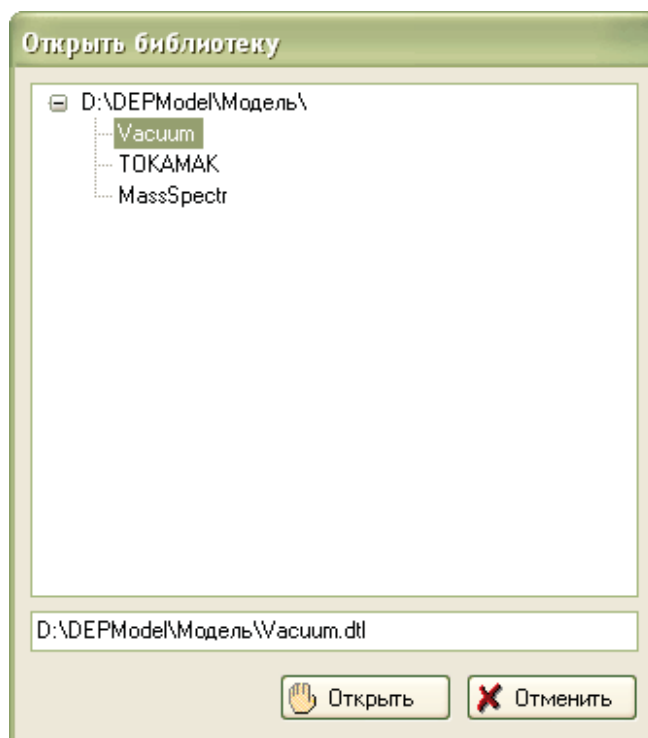



Рис. 1

9.3.4.3 Сохранить библиотеку


Чтобы сохранить библиотеку, надо или нажать на кнопку "Сохранить библиотеку"  или выбрать в главном меню пункт "Библиотека\Сохранить библиотеку" или "Библиотека\Сохранить библиотеку как ...".

При сохранении библиотеки:

- все открытые типы сохраняются в библиотеку;
- сохраняются описания типов (файл <библиотека.dtl>);
- сохраняются [файлы функций](#) (файлы <библиотекаF.cpp>, <библиотекаF.h>).


Если выбран пункт "Сохранить библиотеку как ..." запрашивается имя и расположение новой библиотеки.

9.3.4.4 Проверить библиотеку

При проверке библиотеки, проверяются все описанные в ней типы. Чтобы проверить библиотеку, надо или нажать на кнопку "Проверить библиотеку"  или выбрать в главном меню пункт "Библиотека\Проверить библиотеку".

9.3.4.5 Построить библиотеку

Результатом построения библиотеки являются файлы <библиотека.obj> и файл функций <библиотекаF.obj>.

Чтобы построить библиотеку, надо или нажать на кнопку "Построить библиотеку"  или выбрать в главном меню пункт "Библиотека\Построить библиотеку".

При построении библиотеки:

- выполняется проверка библиотеки. Если во время проверки будут обнаружены ошибки, построение не производится;
- создается файл <библиотека.h>, в котором все описанные в библиотеке типы реализуются в виде описания классов. В описание класса также помещается декларация всех функций, описанных для соответствующего типа в файле функций;
- создается файл <библиотека.c>, в котором описаны конструкторы для всех классов библиотеки;
- файлы <библиотека.c> и <библиотекаF.c> компилируются с помощью C++Builder в объектные файлы <библиотека.obj> и <библиотекаF.obj>.

Важное замечание ! Описание классов в файле <библиотека.h> создается в той последовательности, в какой описаны в библиотеке соответствующие типы. А это значит, что порядок описания типов в библиотеке важен. Если тип "тип А" использует тип "ТипВ", то тип "ТипВ" должен быть описан раньше чем "тип А".

Порядок описания типов в библиотеке легко меняется с помощью процедуры Drag-and-Drop.

9.3.4.6 Файл функций

Пустой файл функций создается автоматически при создании новой библиотеки (рис 1.). В дальнейшем, для каждого типа, описанного в библиотеке, здесь можно определить набор функций.

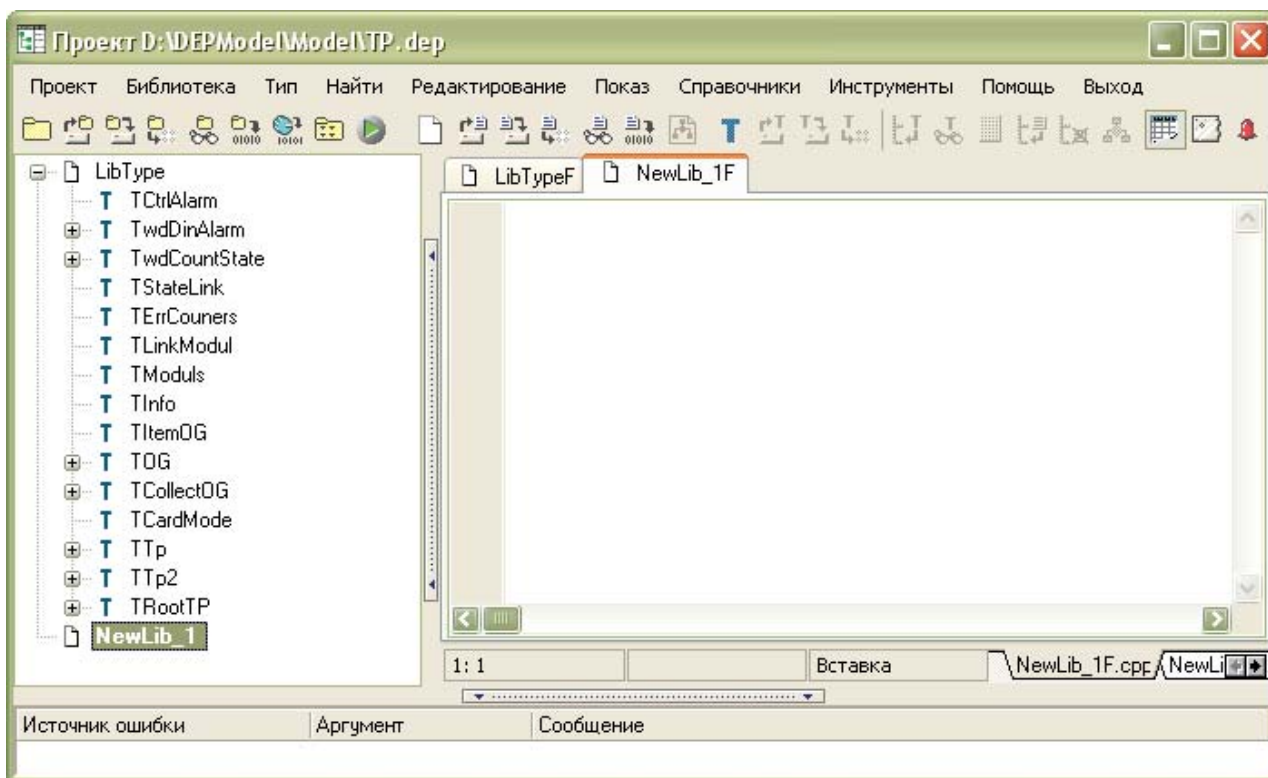


Рис. 1


Для добавления новой функции надо встать на соответствующий тип и нажать на кнопку "Новая функция"  или выбрать пункт меню "Тип\Новая функция". Появится окно, показанное на рисунке 2.



Рис. 2

После нажатия кнопки "Установить" или "Применить" в файле функций появится описание новой функции (рисунок 3).

Обратите внимание на комментарий, начинающийся со строки "--!--". Она указывает на то, что далее идет описание декларации функции. Это описание будет использовано при добавлении декларации функции в файл <библиотека.h>. Если комментарий отсутствует, декларация функции будет создана по умолчанию. В нашем случае это:

с комментарием: **private** virtual void TInfo_func1(int arg1, float arg2);

по умолчанию: **public** void TInfo_func1(int arg1, float arg2);

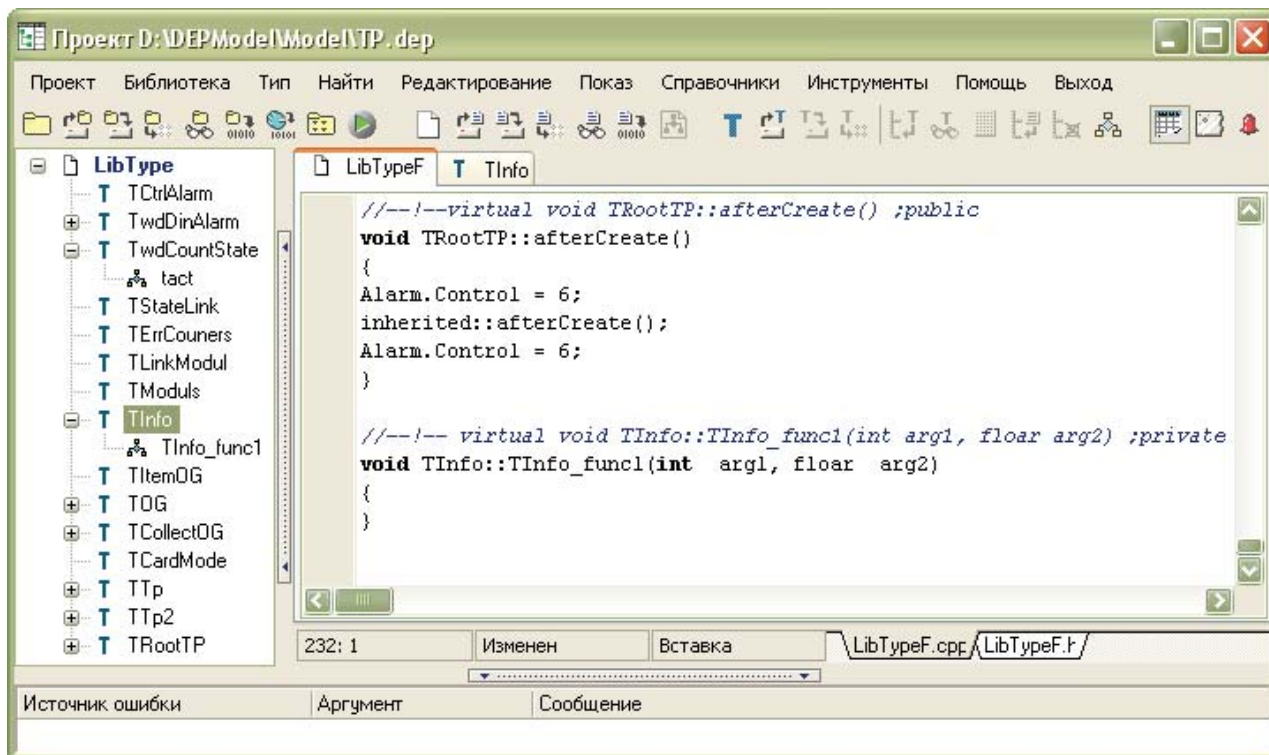


Рис. 3


Новую функцию не обязательно добавлять, как это описано выше. Можно просто набрать текст в окне файла функций.

В общем случае, в файле функций можно писать любой код, определять любые переменные. При построении библиотеки, файл функции будет откомпилирован в том виде, в каком он есть.

9.3.5 Тип

Программа имеет набор базовых типов и функций, которые можно использовать при построении модели. Кроме этого, "Конструктор OPC-модели" позволяет создавать свои типы и наборы функций для них. Тип, который является образом объекта автоматизации, называется **главный тип**.

9.3.5.1 Новый тип

Чтобы добавить в библиотеку новый тип надо или нажать на кнопку "Новый тип"  или выбрать в главном меню пункт "Тип\Новый тип". Появится окно, показанное на рисунке 1.

Для создания нового типа необходимо дать ему имя (русские буквы использовать нельзя) и задать базовый тип. Новый тип наследует все элементы и параметры базового типа (рис. 2).

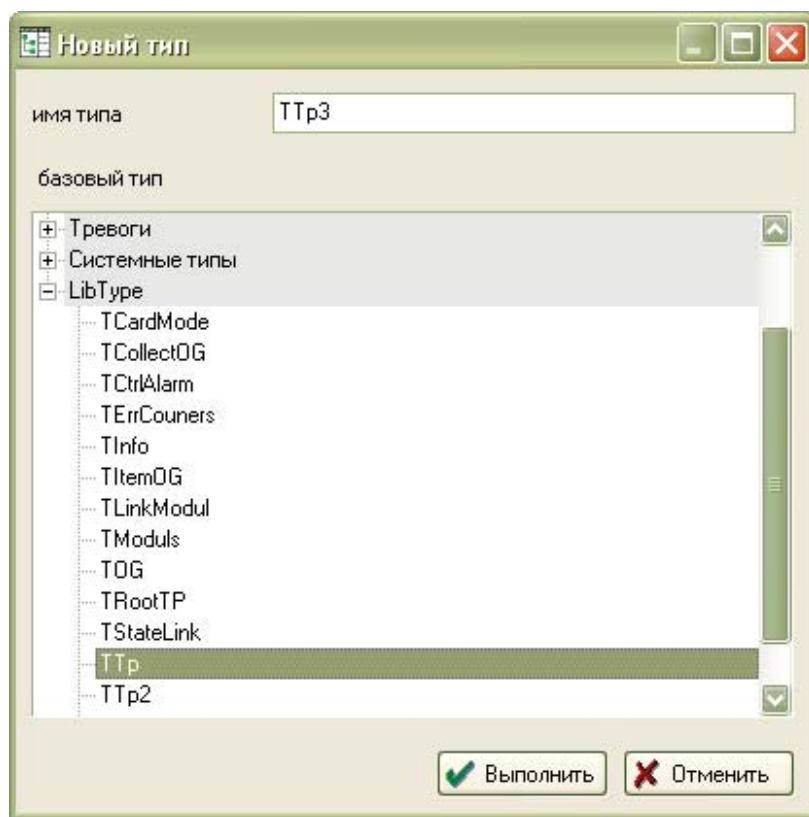


Рис. 1

Созданный тип открывается на редактирование (рисунок 2). Пока он еще живет своей автономной жизнью, в библиотеке его еще нет (это видно, если обратить внимание на список типов библиотеки). Новый тип попадет в библиотеку только после сохранения.

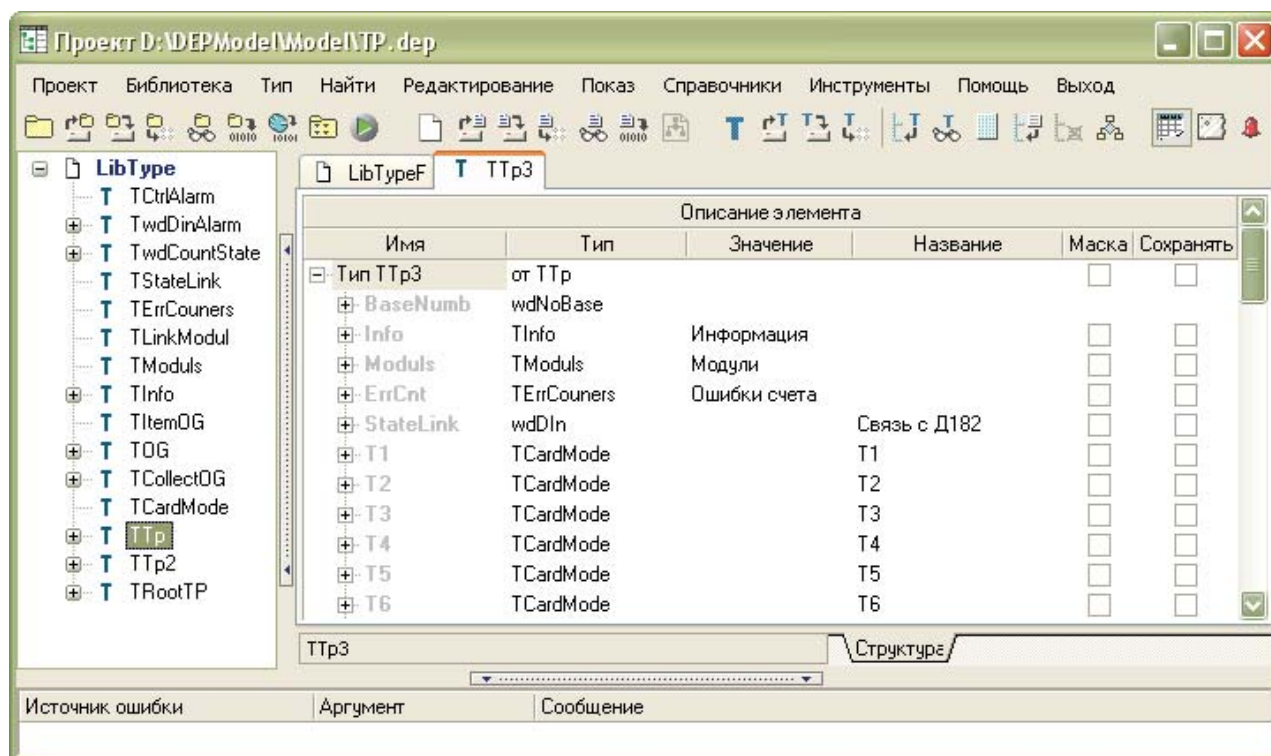



Рис. 2

9.3.5.2 Открыть тип

Чтобы открыть тип надо или нажать на кнопку "Открыть тип"  или выбрать в главном меню пункт "Тип\Открыть тип". В правой части окна появляется соответствующая закладка (рисунок 1).

Когда тип открывается на редактирование, создается его копия и редактируется именно она. В библиотеке пока еще никакие изменения не происходят. Все измененный тип попадет в библиотеку только после сохранения.

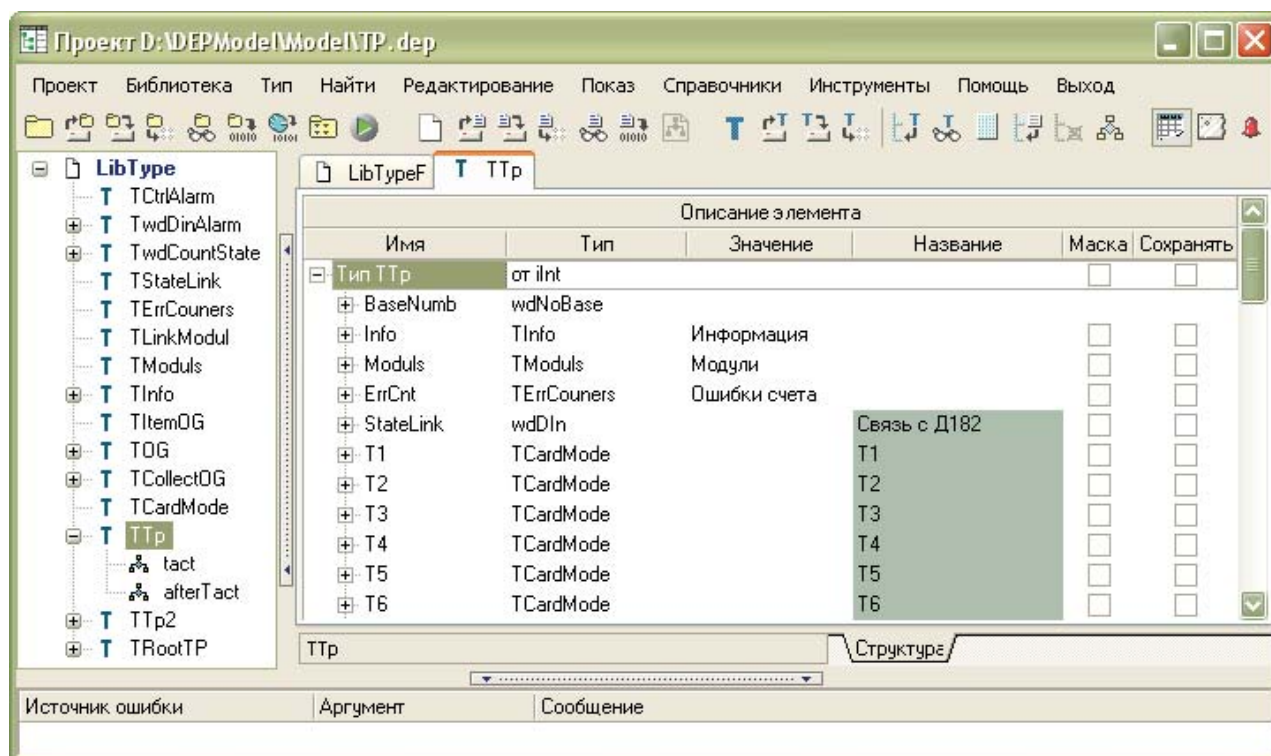


Рис. 1

9.3.5.3 Сохранить тип

Чтобы сохранить тип надо или нажать на кнопку "Сохранить тип"  или выбрать в главном меню пункт "Тип\Сохранить тип" или "Тип\Сохранить тип как".

Любое редактирование типа происходит вне библиотеки. Если тип только что создан, в библиотеке его пока еще нет вообще. Если тип открывается на редактирование, создается его копия и именно в ней производятся все изменения.

Чтобы новый тип попал в библиотеку или чтобы в ней отразились результаты редактирования уже существующего типа, его надо сохранить.

Если выбрано "Сохранить тип как", то в библиотеке остается старая копия, и будет запрошено имя, под которым надо сохранить новую копию.

9.3.5.4 Удалить тип

Чтобы удалить тип из библиотеки надо указать его в дереве типов и выбрать пункт меню "Тип\Удалить тип" (рисунок 1). Перед удалением типа надо убедиться, что он не используется в проекте.

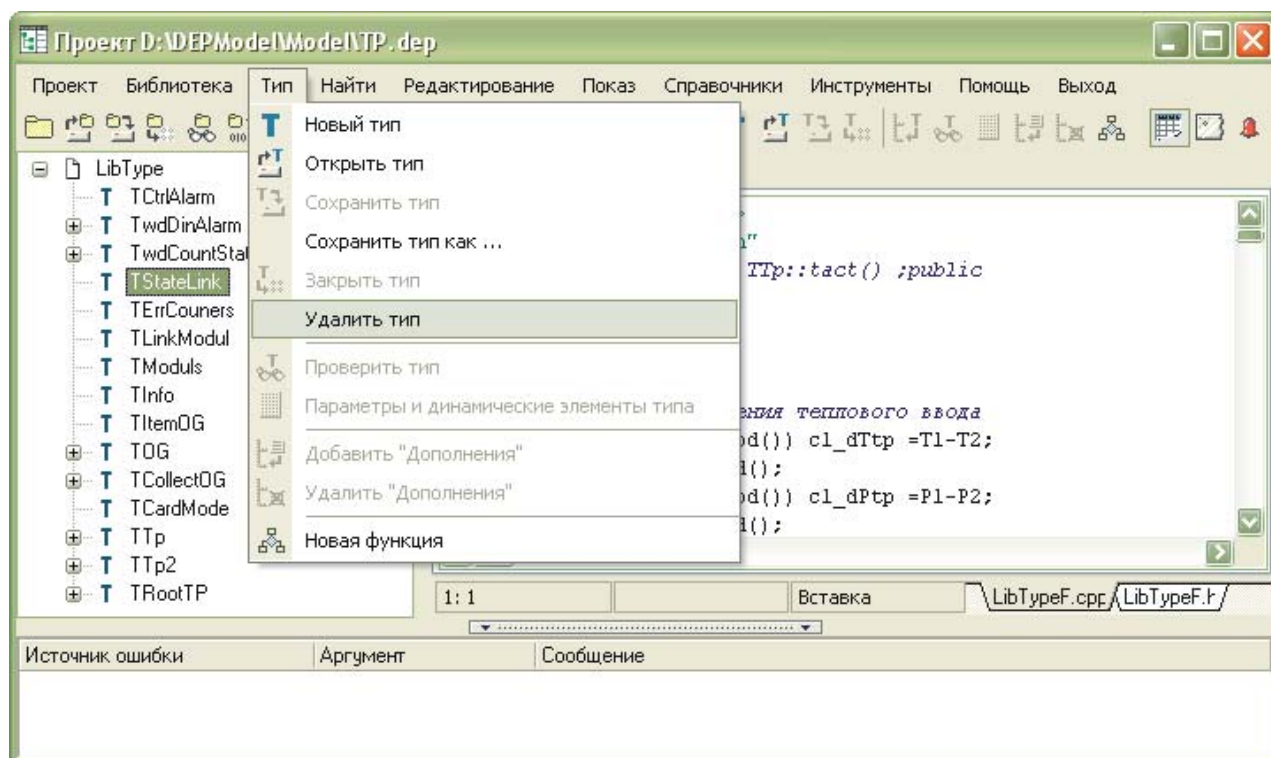



Рис. 1

9.3.5.5 Дополнения типа

При построении библиотеки каждый тип реализуется в виде класса. В декларации класса (в файле <библиотека.h>) перечисляются все элементы типа. Если необходимо добавить в описание класса что-то еще, это можно сделать с помощью раздела "Дополнения". Все что описано в разделе "Дополнения" данного типа вставляется в его декларацию.

При создании нового типа, раздел "Дополнения" автоматически не создается.

Чтобы добавить раздел "Дополнения" надо или нажать на кнопку "Добавить "Дополнения""  или выбрать в главном меню пункт "Тип\Добавить "Дополнения". Появляется соответствующая закладка (рисунок 1). Все, что будет здесь написано, попадет в декларацию класса.

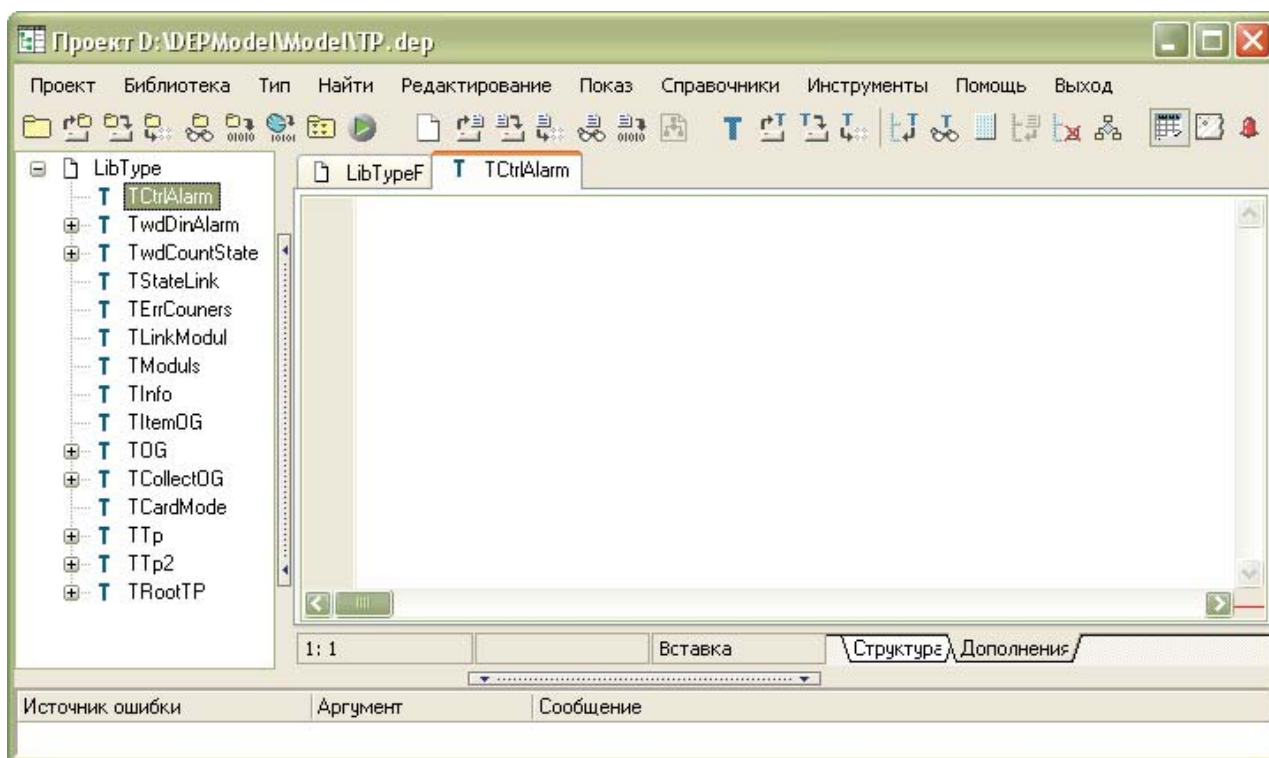


Рис. 1

Чтобы удалить раздел "Дополнения" надо или нажать на кнопку "Удалить "Дополнения""  или выбрать в главном меню пункт "Тип\Удалить "Дополнения".

9.3.5.6 Изменить имя типа

Для изменения имени типа надо в диалоговом окне указать его новое имя (рисунок 1).

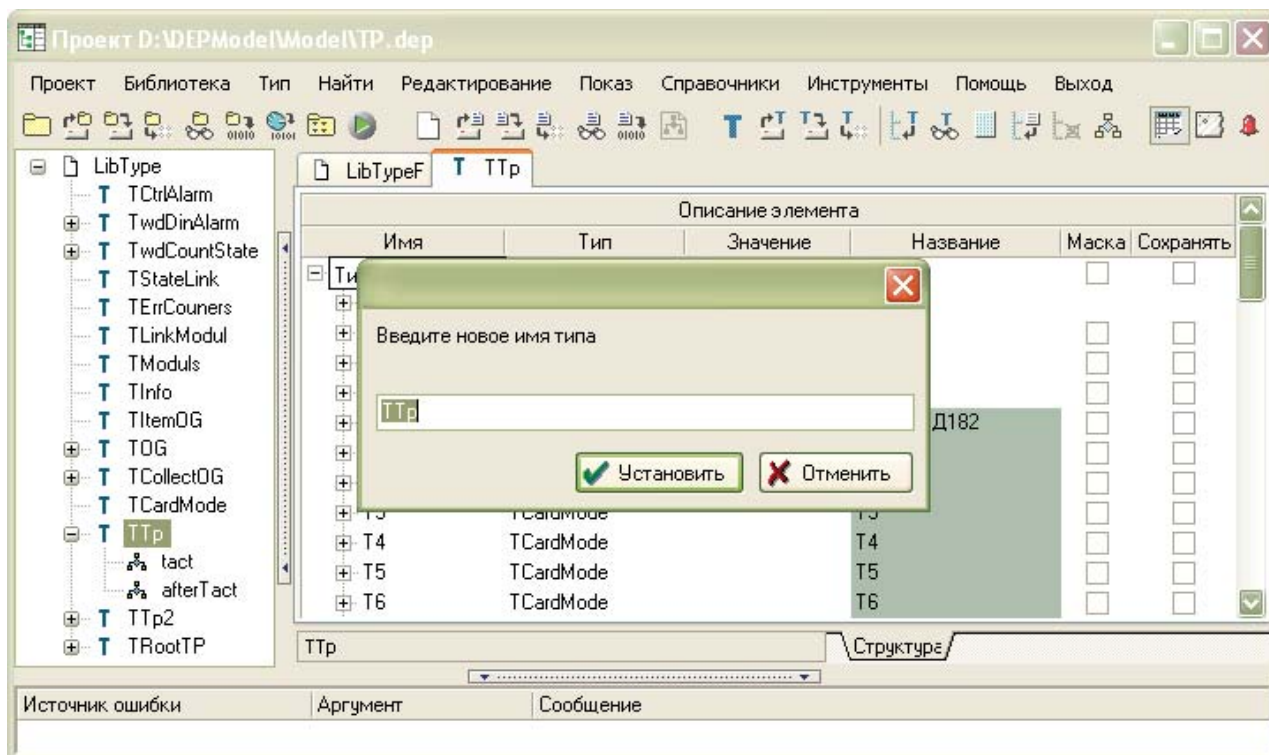


Рис. 1

"Конструктор модели" проверит новое имя на допустимость, а также просмотрит известные ему библиотеки (см. [установки проекта](#)) и проверит, что новое имя типа нигде не встречается, в противном случае будет сохранено старое название. Но это не уберезет от возможных ошибок. Это имя может встречаться в другой библиотеке, неизвестной данному проекту. В этом случае возможна ситуация, когда в каком-то проекте, две библиотеки с одинаково названными типами встретятся, и проект невозможно будет нормально открыть.

Менее фатальна ситуация, когда тип где-то используется, а его взяли и переименовали. В этом случае всем элементам старого типа надо будет присвоить новый тип (см. [изменить тип элемента](#)).

Из сказанного выше видно, что переименование типа, процедура неприятная и ее желательно избегать.

Лучше еще на этапе создания типа, хорошо подумать, как его назвать.

9.3.5.7 Изменить базовый тип

Для изменения базового типа надо выбрать новый базовый тип в диалоговом окне (рисунок 1). Так как тип полностью наследует структуру базового типа, после его изменения, в структуре редактируемого типа могут появиться новые элементы, а какие-то пропадут. При изменении базового типа возможна ситуация, когда какие-то параметры останутся легальными. Например, если в новом базовом типе есть элементы с теми же именами и такой же иерархией. Но они могут иметь другой смысл и для них могут потребоваться другие значения параметров. Если при изменении базового типа выбрать опцию "с сохранением параметров", то все легальные параметры будут сохранены. В противном случае они будут удалены.

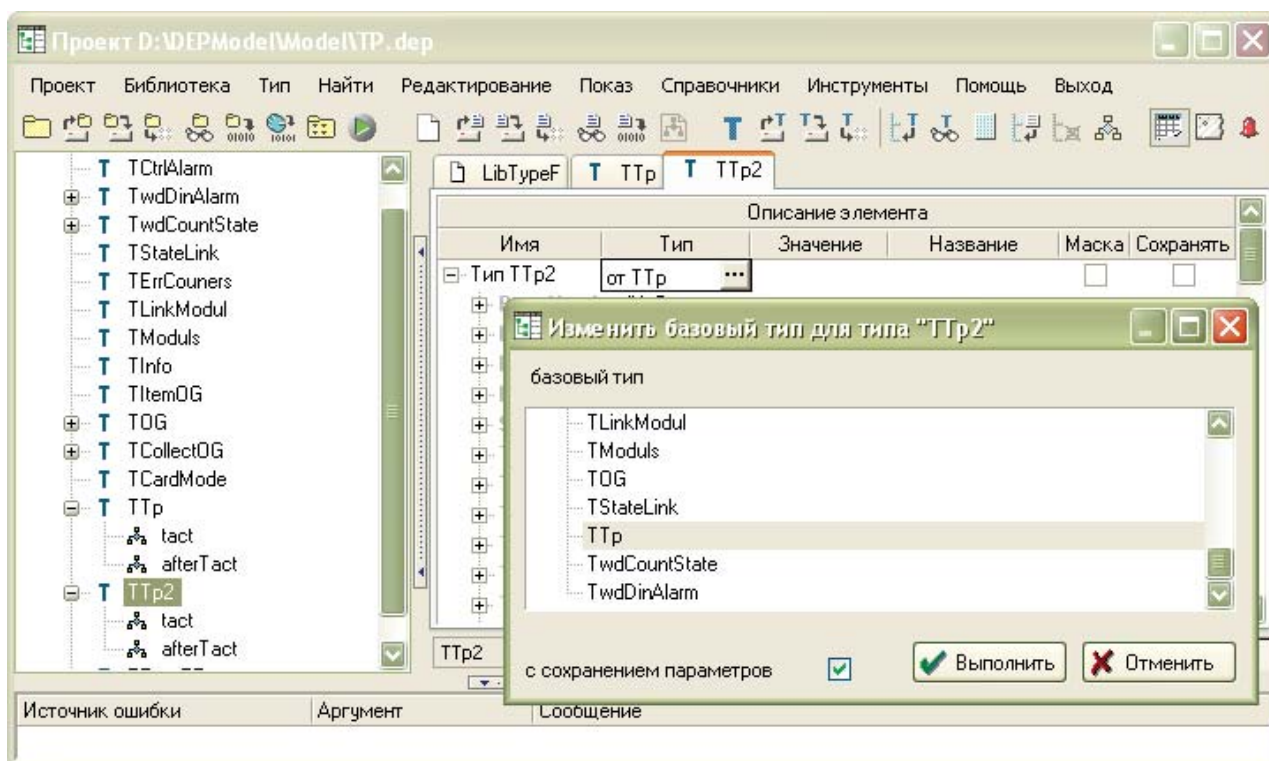



Рис. 1

9.3.5.8 Новый элемент

Для того чтобы добавить новый элемент надо или нажать на кнопку "Новый элемент"  или выбрать в главном меню пункт "Редактирование\Новый элемент". Новый элемент всегда добавляется после текущего выбранного элемента.

Появится окно, показанное на рисунке 1.

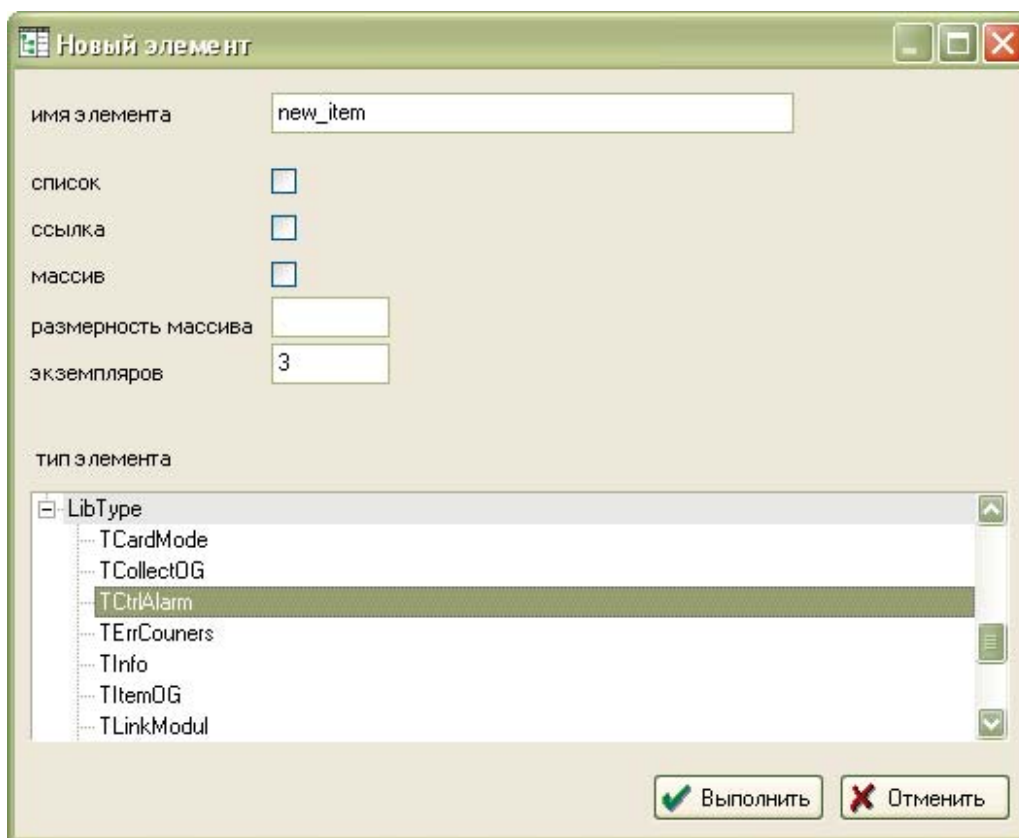


Рис. 1

Надо обязательно указать имя и тип нового элемента. Кроме того, его можно наделить дополнительными свойствами:

- это может быть список элементов указанного типа;
- это может быть ссылка на элемент указанного типа;
- это может быть массив элементов указанного типа, в этом случае надо указать его размерность;
- можно добавить не один, а сразу несколько элементов, указанного типа (рисунок 2).

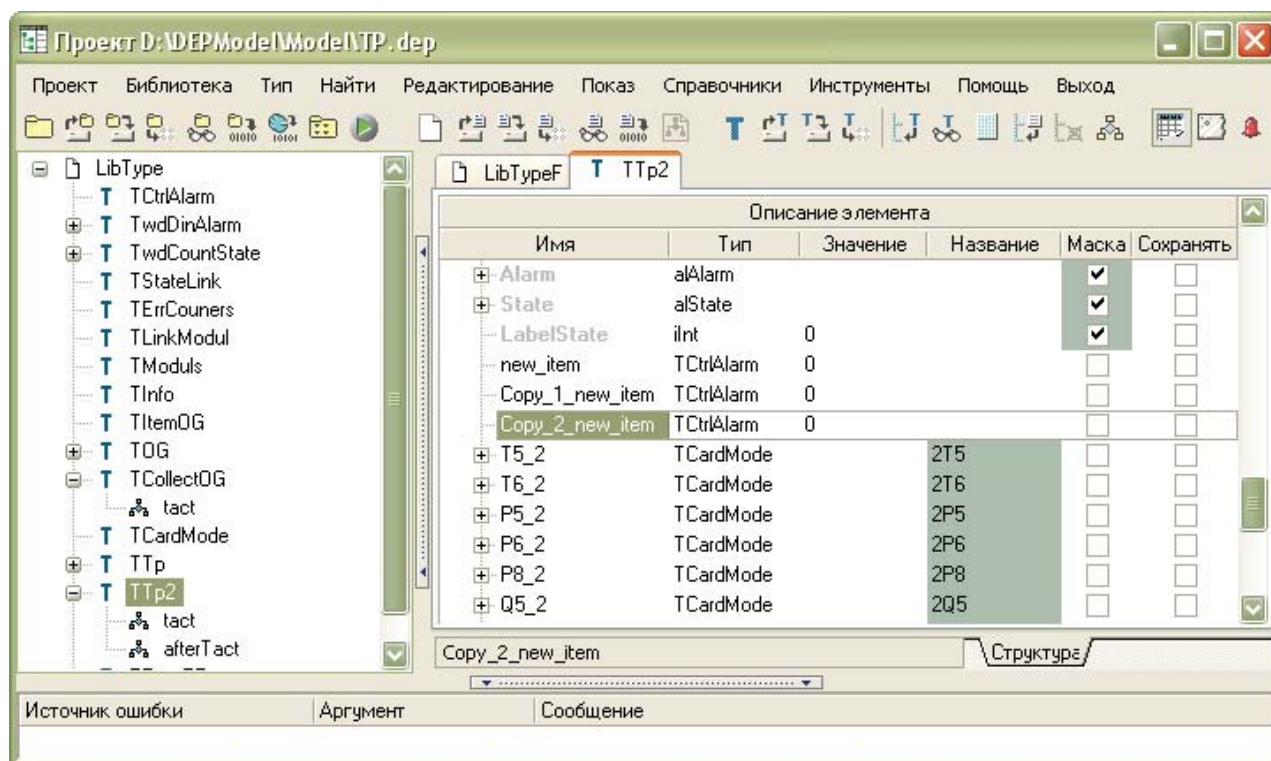


Рис. 2

9.3.5.9 Динамические элементы

Динамическими называются элементы, которые именуются относительно других элементов или, другими словами это элементы, добавленные НЕ на первый уровень иерархии. Добавить динамический элемент можно через пункт меню "Редактирование\Новый динамический элемент".

Рассмотрим это на примере.

На рисунке 1 показан тип "сех", который имеет элемент "yzel1" типа "yzel". Элемент "yzel1", в свою очередь имеет элементы "device1" и "device2", их полные имена относительно типа "сех" - "yzel1\device1" и "yzel1\device2".

Теперь встанем на "yzel1" и добавим динамический элемент типа "device", назвав его "device3".

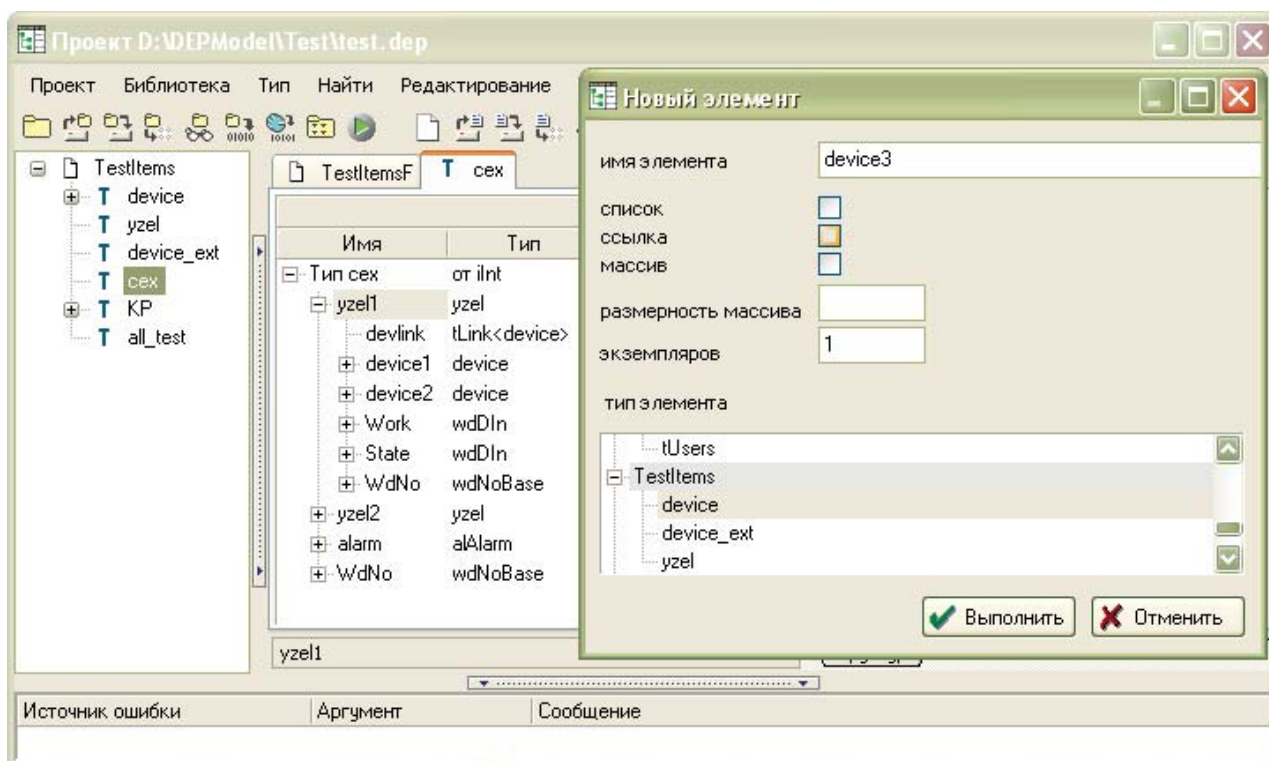


Рис. 1

Результат этих действий представлен на рисунке 2. Появился элемент "yzel1\device3". Создается иллюзия, будто в типе "yzel" узел присутствует элемент "device3", хотя на самом деле его там нет. Или можно сказать, что тип "сех" создал в элементе "yzel1" динамический элемент "device3".

Динамические элементы выделяются синим жирным шрифтом.

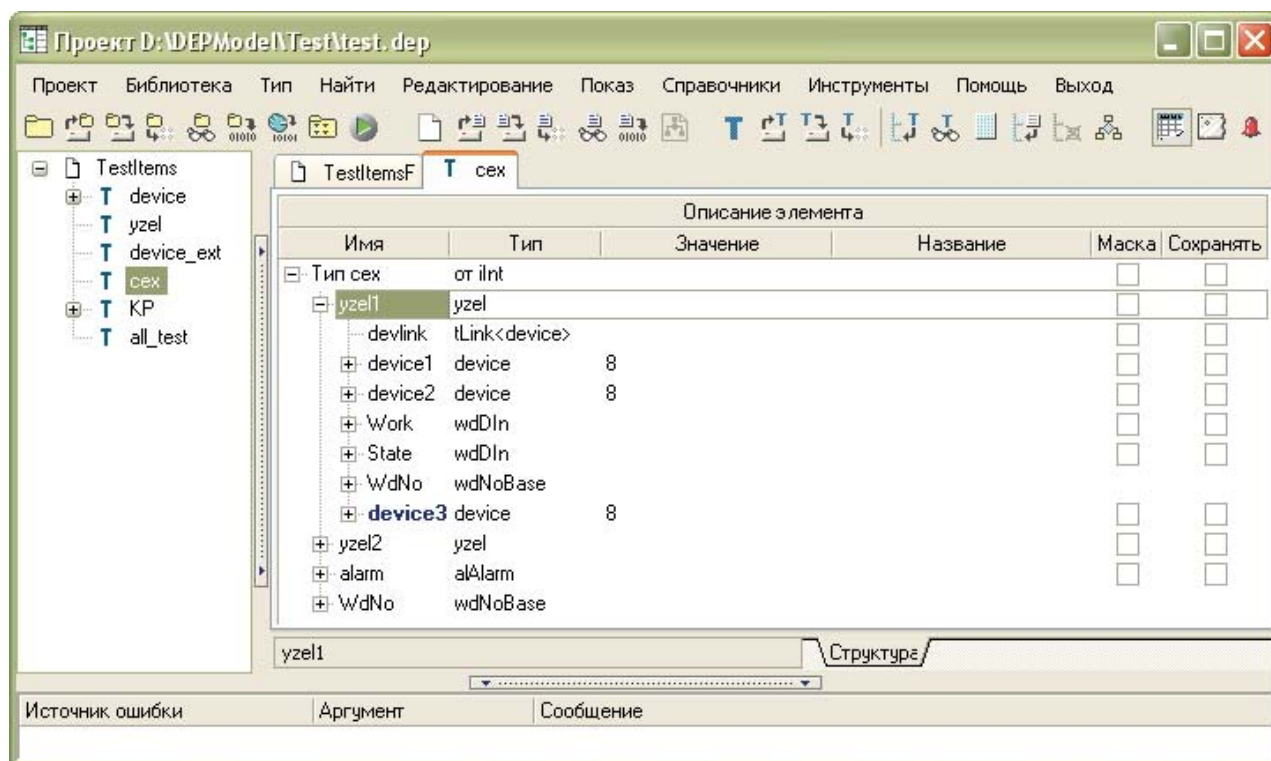


Рис. 2

9.3.5.10 Удалить элемент

Удалить можно только элементы, принадлежащие данному типу.

В примере, приведенном на рисунке 1, из типа "sex" можно удалить элемент "yzel1". Элемент "device1" удалить нельзя, т.к. он принадлежит типу "yzel", чтобы его удалить, надо открыть на редактирование именно тип "yzel". А вот [динамический элемент](#) "device3" удалить можно, т.к. несмотря на иллюзию, что он принадлежит типу "yzel", он был добавлен при редактировании типа "sex" и принадлежит именно ему.

Чтобы удалить выбранный элемент, надо выбрать пункт меню "Редактирование\Удалить элемент".

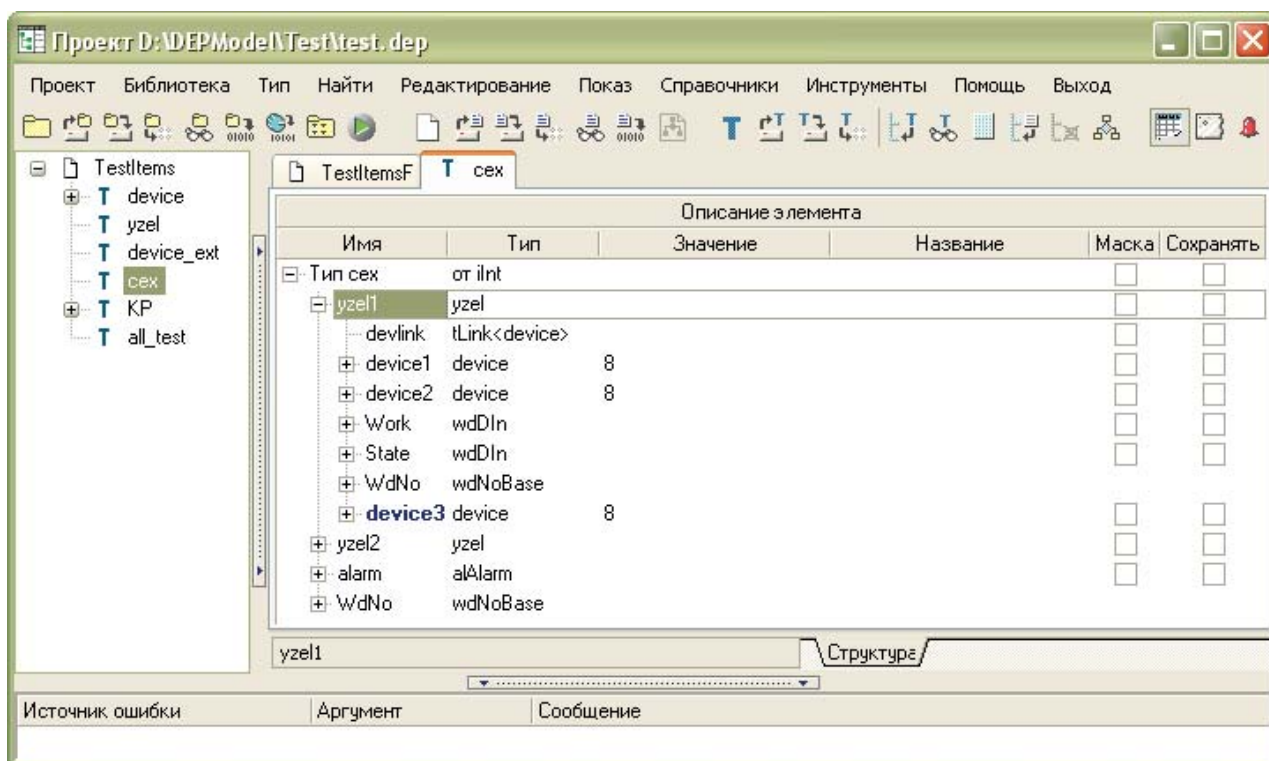


Рис. 1

9.3.5.11 Переименовать элемент

Переименовать можно только элементы, принадлежащие данному типу. Чтобы переименовать элемент, надо просто ввести его новое имя.

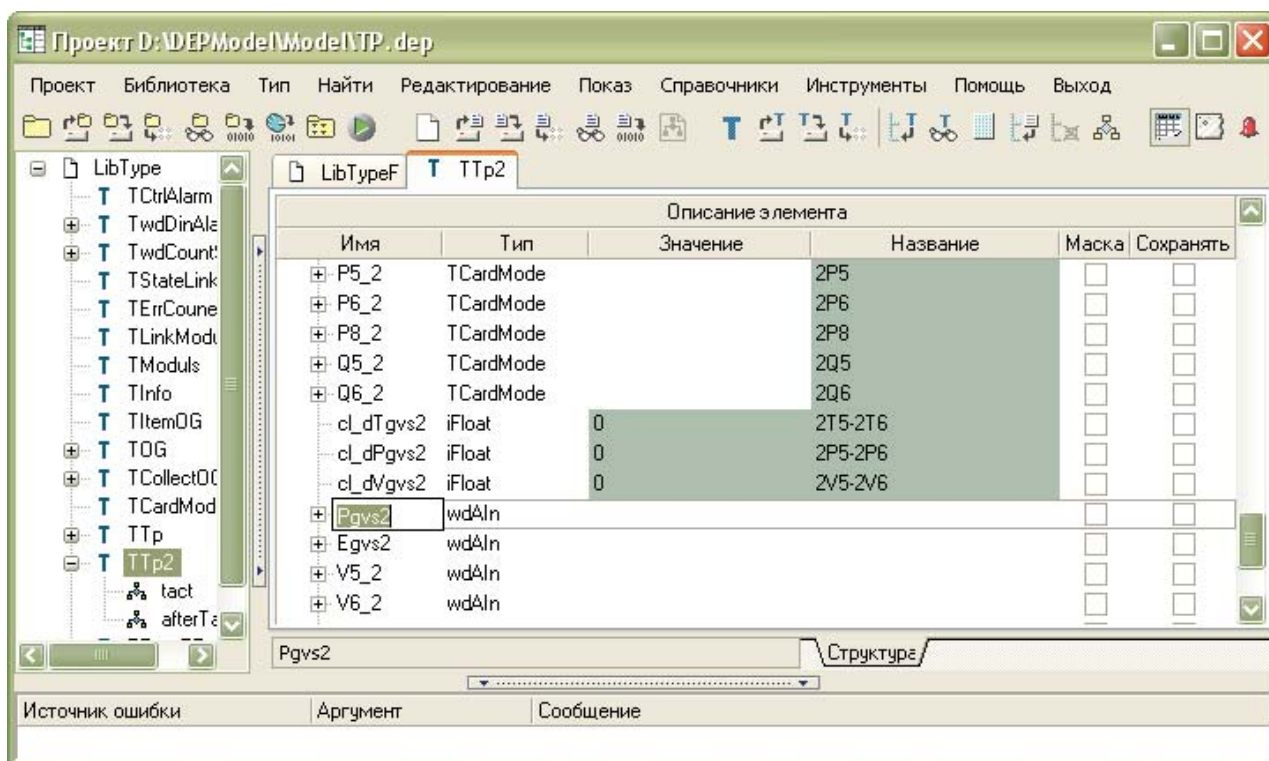


Рис. 1

9.3.5.12 Изменить тип элемента

Изменить тип можно только элементам, принадлежащим данному типу. Для этого надо просто выбрать новый тип элемента.

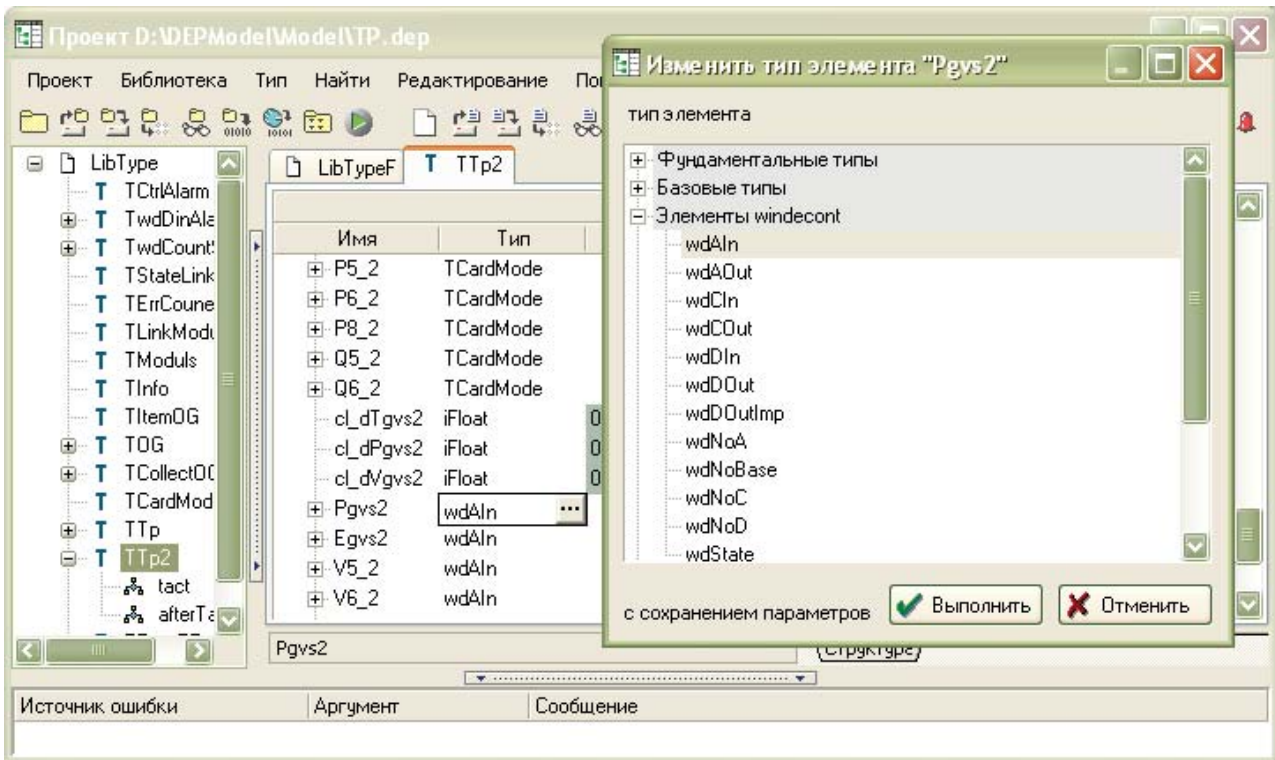


Рис. 1

9.3.5.13 Изменить размерность элемента

Изменить размерность можно только элементам, принадлежащим данному типу. Элемент может быть единичным элементом или массивом. Для этого надо воспользоваться элементом меню "Редактирование\Массив".

В диалоговом окне надо сказать является элемент массивом или нет и для массива указать его размерность.

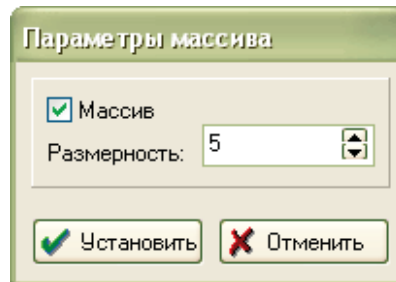


Рис. 1

9.3.5.14 Фильтр строк

При редактировании типа, есть возможность отфильтровать элементы, которые являются номерами дискретов, аналогов или счетчиков (рисунок 1). Делается это через пункты меню "Показ\Фильтр строк\Номера дискретов", "Показ\Фильтр строк\Номера аналогов", "Показ\Фильтр строк\Номера счетчиков". При фильтрации показываются только элементы, которые являются номерами в соответствующей базе и их "родители". На рисунке 1 показана структура типа "TRootTP" без фильтрации, на рисунке 2 она же с фильтром по номерам дискретов. Установка фильтра снимает предыдущий фильтр. При установке/снятии фильтра важно, какой элемент является текущим. Фильтр распространяется только на текущий элемент со всеми его подэлементами.

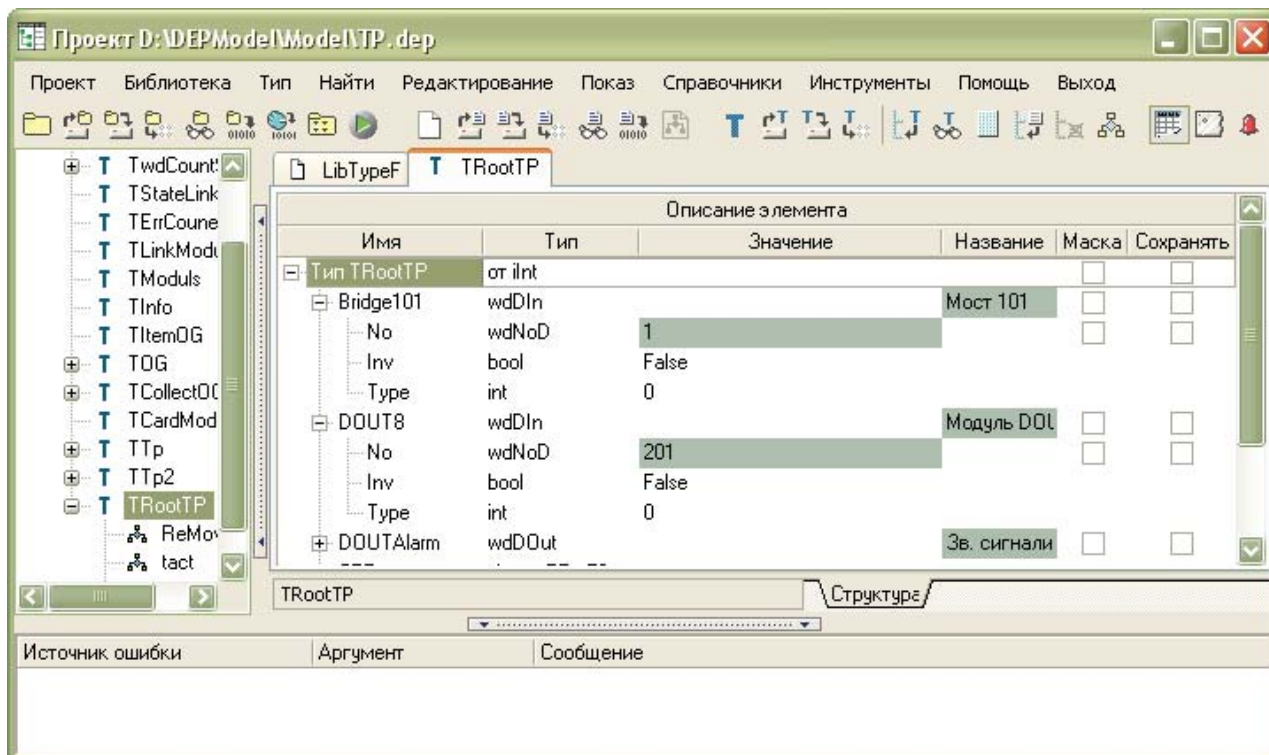


Рис. 1

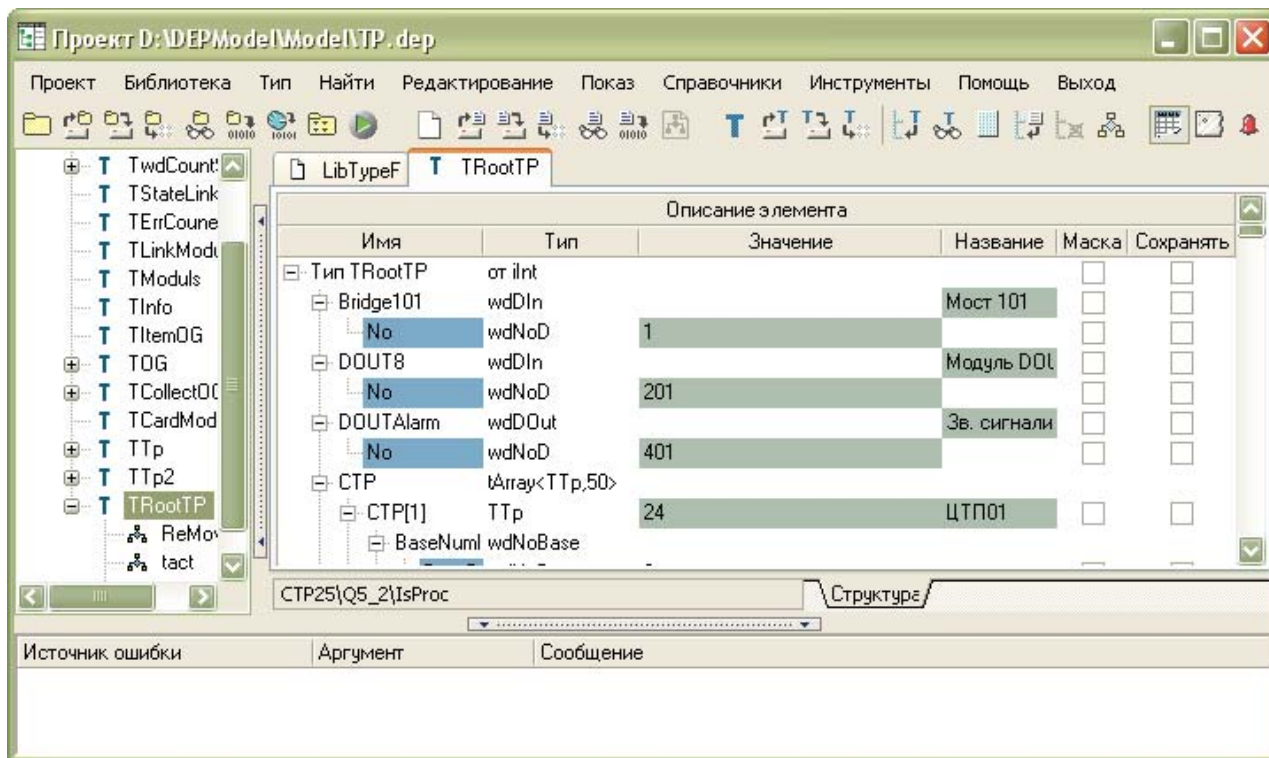


Рис. 2

9.3.5.15 Параметры

Любой тип и любой его элемент (как экземпляр какого-то типа) имеет ряд параметров. Какие конкретно параметры, зависит от основного базового типа, от которого произошли предки указанного типа. Другими словами, если тип "Тип А" произошел от типа "Тип В", а тип "Тип В" от типа "Тип С", а тип "Тип С" от основного типа "iInt", то все перечисленные типы будут иметь одинаковый набор параметров и определяться он будет типом "iInt". В общем случае, можно считать, что все элементы имеют одинаковый набор параметров, отличия невелики.

Массивы, списки и ссылки имеют усеченный набор параметров.

"Конструктор OPC-модели" показывает все параметры, какие только могут быть. Если у элемента конкретный параметр отсутствует, в соответствующее поле просто ничего нельзя ввести.

При создании типа, он наследует значения всех параметров от базового типа. В дальнейшем, при редактировании типа, для любого элемента (на любом уровне иерархии) можно изменить значение любого существующего параметра. Но это изменение будет действовать только в рамках данного типа.

Если значение параметра определено в данном типе, соответствующее поле выделено серо-зеленым цветом. Если значение параметра определено в базовом типе или в типе, к которому относится элемент (в этом случае оно называется значением по умолчанию), соответствующее поле имеет стандартный цвет фона.

Как только при редактировании типа что-то ввели в поле параметра, даже если там было то же самое значение, значение параметра будет считаться определенным в данном типе. Чтобы вернуться к значению по умолчанию, надо нажать правую кнопку мышки в поле параметра и в появившемся меню выбрать пункт "По умолчанию".

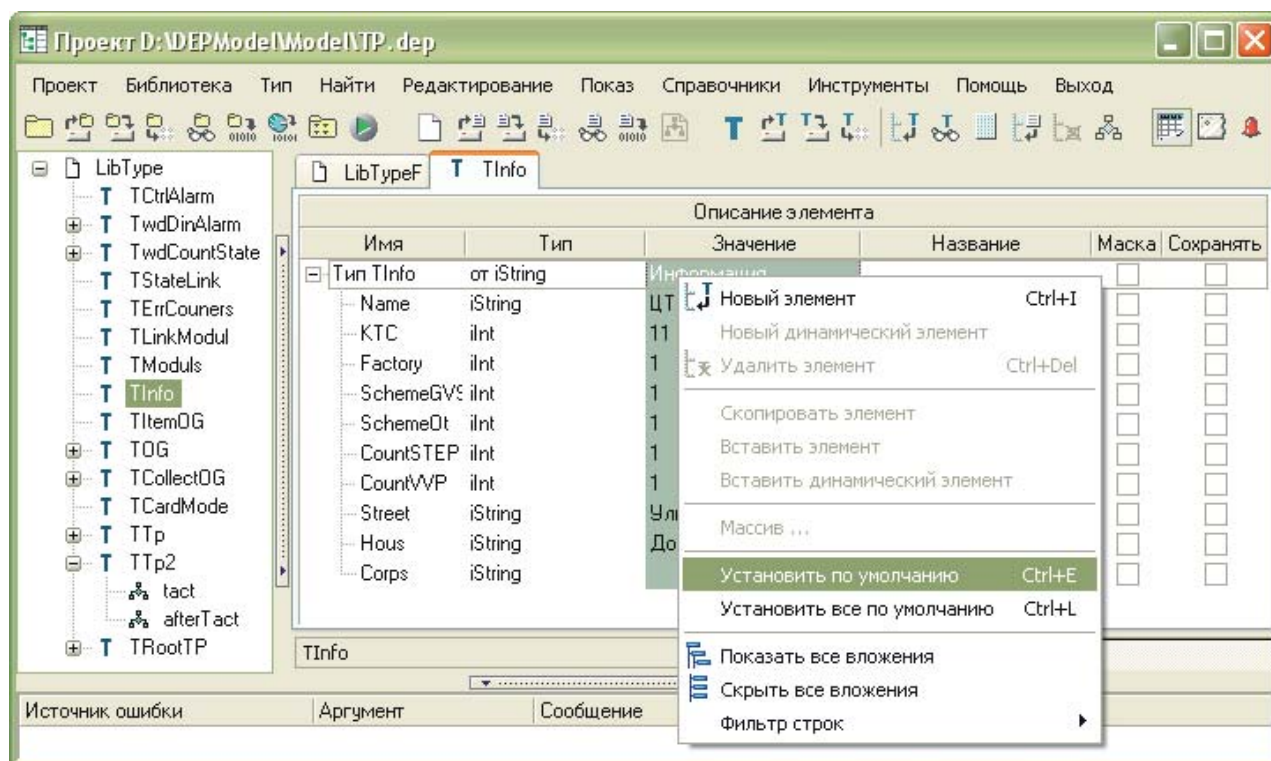


Рис. 1

Полный набор параметров следующий:

- Значение - значение элемента, зависит от его типа;
- Название - название элемента. Русские названия допустимы. Под этим именем элементы архивируются;
- Маска - если элемент замаскирован, это равносильно тому, что его просто нет;
- Сохранять - если параметр установлен, текущее значение элемента запоминается и восстанавливается при рестарте модели. Обычно используется для различных уставок;
- Д - признак того, что изменение значения элемента надо запоминать в архиве событий;
- А - признак того, что изменение значения элемента надо запоминать в архиве аналогов;
- С - признак того, что изменение значения элемента надо запоминать в архиве счетчиков;
- АС - признак того, что усредненное значение элемента надо запоминать в архиве аналогов;
- АМ - признак того, что мгновенное значение элемента надо запоминать в архиве аналогов;
- АМн - признак того, что минимальное значение элемента надо запоминать в архиве аналогов;
- АМх - признак того, что максимальное значение элемента надо запоминать в архиве аналогов;
- СМ - признак того, что мгновенное значение элемента надо запоминать в архиве счетчиков;
- СР - признак того, что приращение значения элемента надо запоминать в архиве счетчиков;
- Загрубление счетчиков (для А и С); - определяет, насколько должно измениться значение, чтобы его поместили в архив аналогов или счетчиков значения элемента;
- Таблица кодировки - имя таблицы из справочника "[Кодировка дискретов](#)". Используется при показе значения (текущего или в архиве) дискрета;
- Таблица периодов - имя таблицы из справочника "[Периоды архивирования](#)". Определяет периодичность сохранения в архивы аналогов и счетчиков значения элемента;
- Важность - значение важности;
- Название важности - название важности из справочника "[События оперативного журнала](#)";
- Квитировать - признак того, что надо квитировать событие в оперативном журнале.

9.3.5.16 Параметр ЗНАЧЕНИЕ номеров в базе контроллера Windeccont

В параметре "Значение" элементов wdNoD(номер дискрета), wdNoA(номер аналога) и wdNoC(номер счетчика) кодируется номер в базе контроллера Windeccont. Это можно сделать тремя способами:

- указать абсолютный номер в базе параметров;
- указать смещение относительно текущего номера. Текущим является ближайший номер вверх по иерархии;
- указать смещение относительно указанного номера. В этом случае надо также указать относительно кого (база).

На рисунке 1 показано окно, которое появится при редактировании поля "Значение" для номеров в базе контроллера Windeccont.

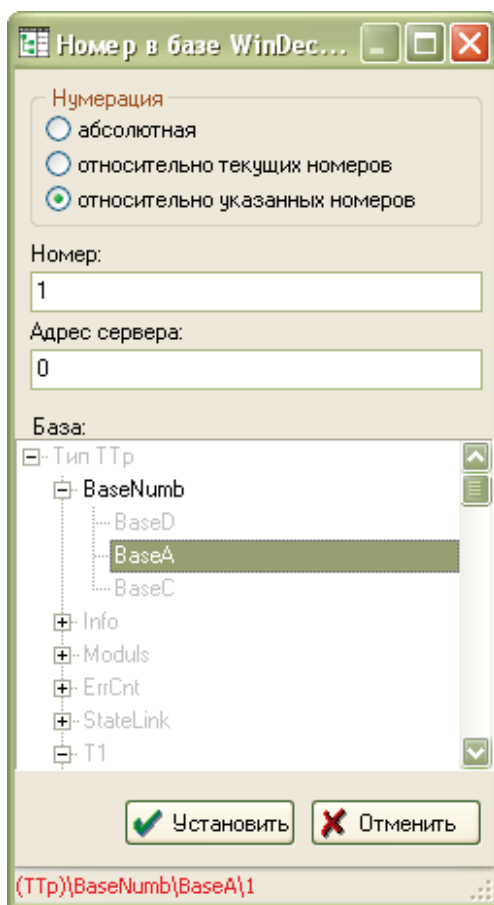


Рис. 1

9.3.5.17 Параметр ЗНАЧЕНИЕ элементов ССЫЛКА

Значением элемента "Ссылка" является строка, состоящая из имени типа, при редактировании которого это значение определяется и полного имени элемента, на который ссылается элемент "ссылка". Сослаться можно только на элемент, имеющий тот же тип, что и элемент "ссылка".

В примере, приведенном на рисунке 1, элемент "сех[1]\yzel1\devlink" является ссылкой на элемент "сех[1]\yzel1\device1" и ее значение определяется при редактировании типа "КР". Поэтому значением ссылки является строка "(КР)\сех[1]\yzel1\device1".

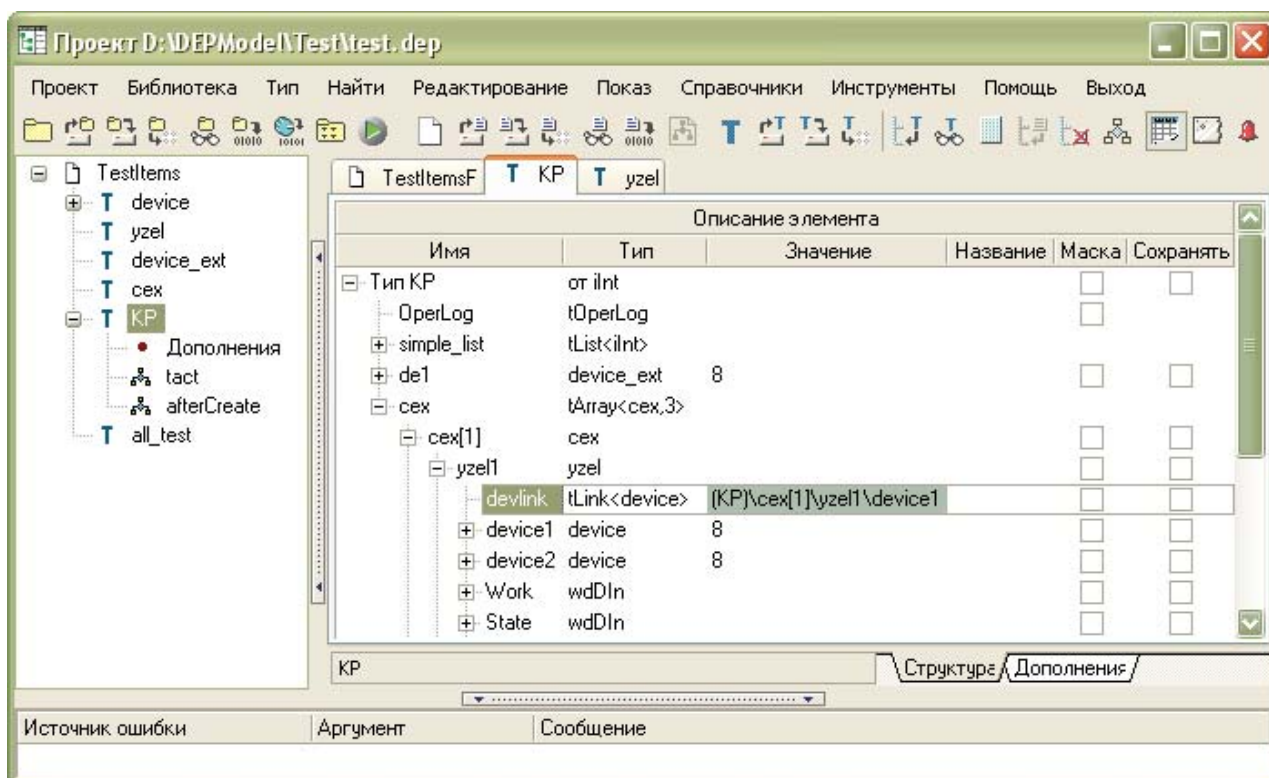



Рис. 1

На рисунке 2 показано окно, которое появится при редактировании параметра "Значение" для элемента "Ссылка".



Рис. 2

9.3.5.18 Просмотр параметров и динамических элементов

Чтобы увидеть все [параметры](#) и все [динамические элементы](#) типа надо или нажать на кнопку "Параметры и динамические элементы типа"  или выбрать пункт меню "Тип\Параметры и динамические элементы типа".

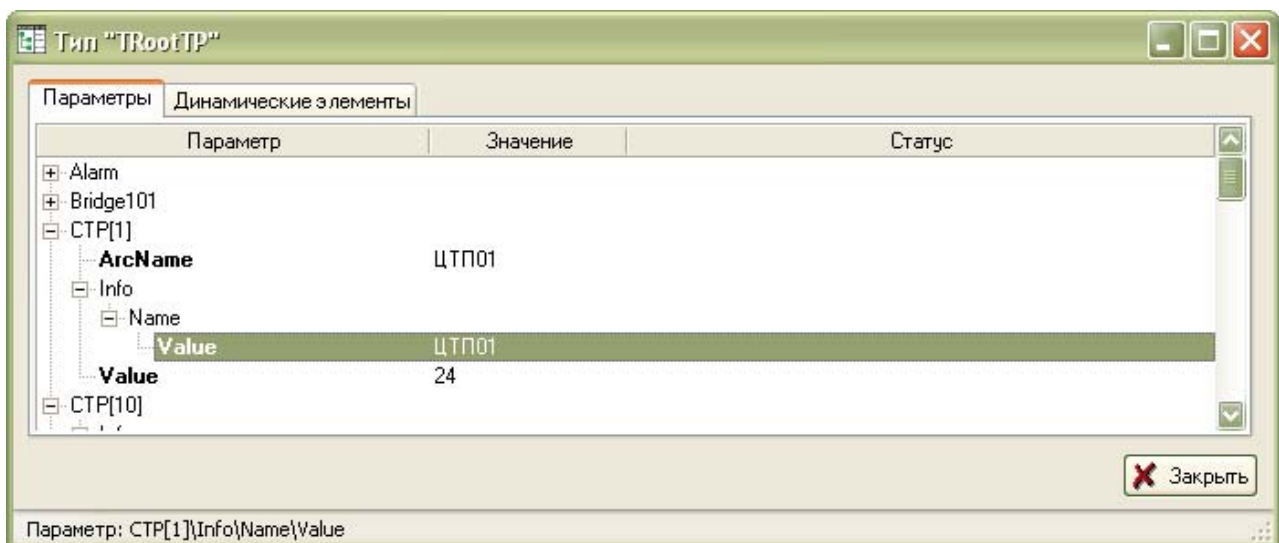


Рис. 1

Рассмотрим пример, как значение параметра может стать некорректным и как с этим бороться. На рисунке 2 отображена структура типа "TTP", где есть элемент Info.

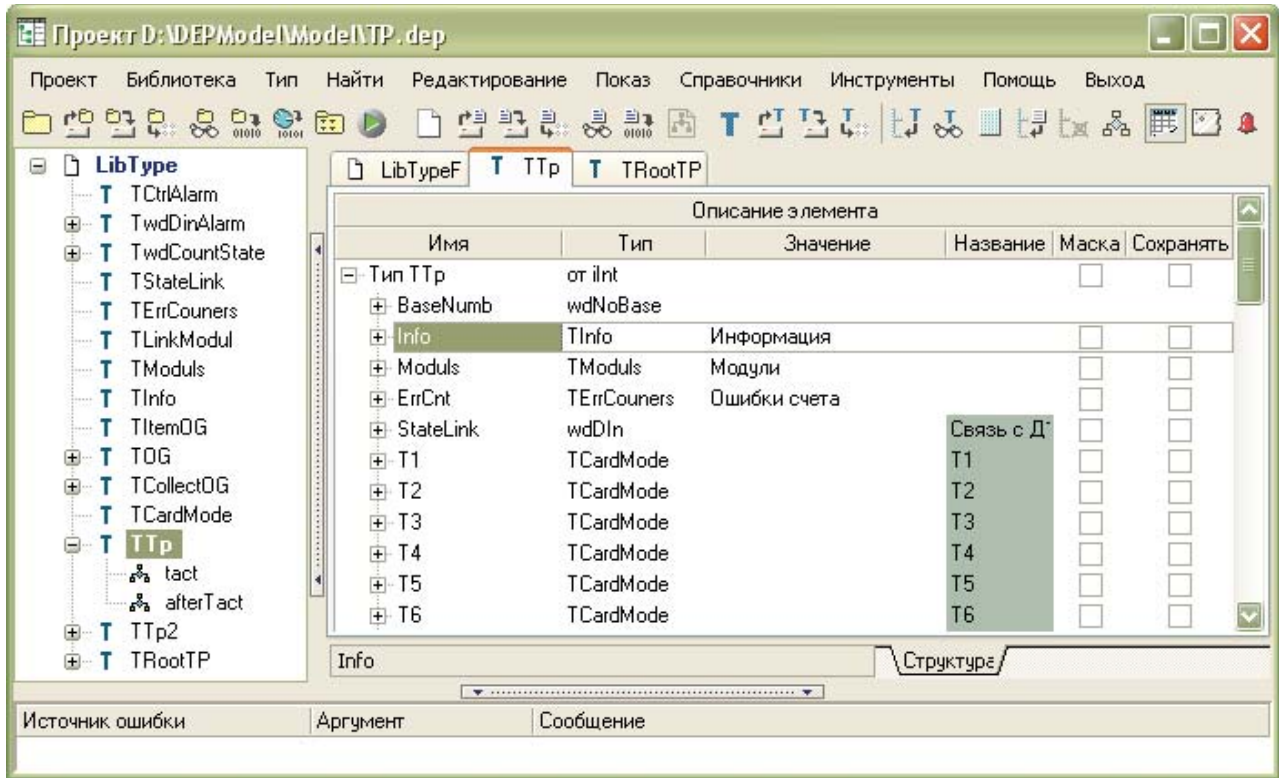


Рис. 2

На рисунке 3 отображена структура типа "TRootTP", который содержит 50 элементов типа "TTP" и где для каждого из этих элементов, определено значение поля "Info\Name\Value". Другими словами, при описании типа "TRootTP" определены значения параметров "CTP[1]\Info\Name\Value", "CTP[2]\Info\Name\Value" ... "CTP[50]\Info\Name\Value" (рисунок 1).

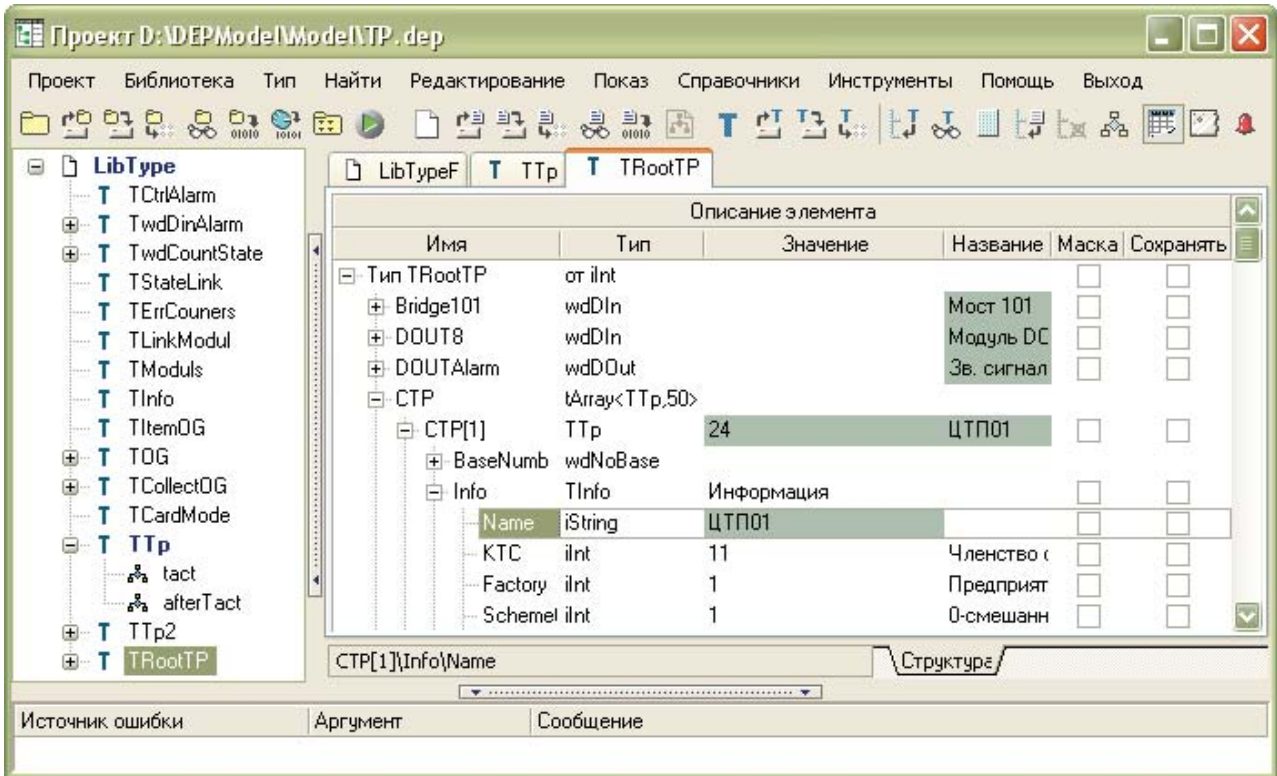


Рис. 3

Если теперь в типе "TTr" переименовать элемент "Info", или в типе "TInfo" переименовать элемент "Name", параметры "CTP[1]\Info\Name\Value", "CTP[2]\Info\Name\Value" ... "CTP[50]\Info\Name\Value" окажутся некорректными. В поле "статус" (рисунок 4) будет сообщение об ошибке.

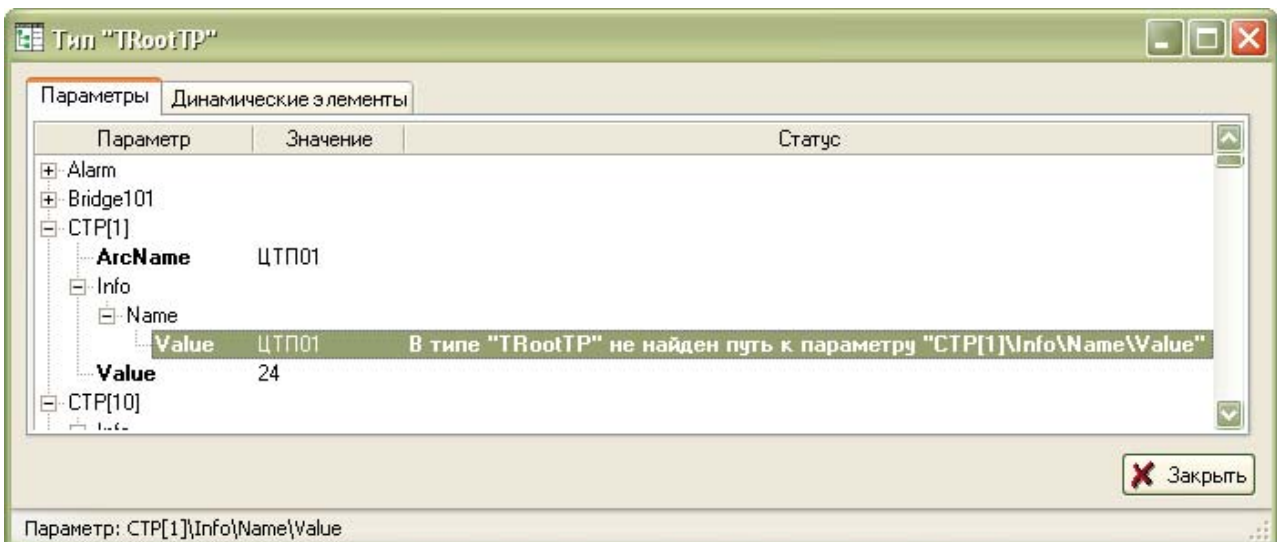


Рис. 4

Возможны два варианта решения проблемы. Или удалить параметр, или переименовать. При нажатии на правую кнопку мышки появляется контекстное меню (рисунок 5). Если, как в нашем примере, проблема возникла из-за того, что в типе "TTr"

элемент "Info" переименовали, например, в "Information", надо встать на элемент "Info", нажать правую кнопку мышки, в появившемся меню выбрать пункт "переименовать" (рисунок 5) и затем в диалоговом окне ввести новое название элемента "Information".

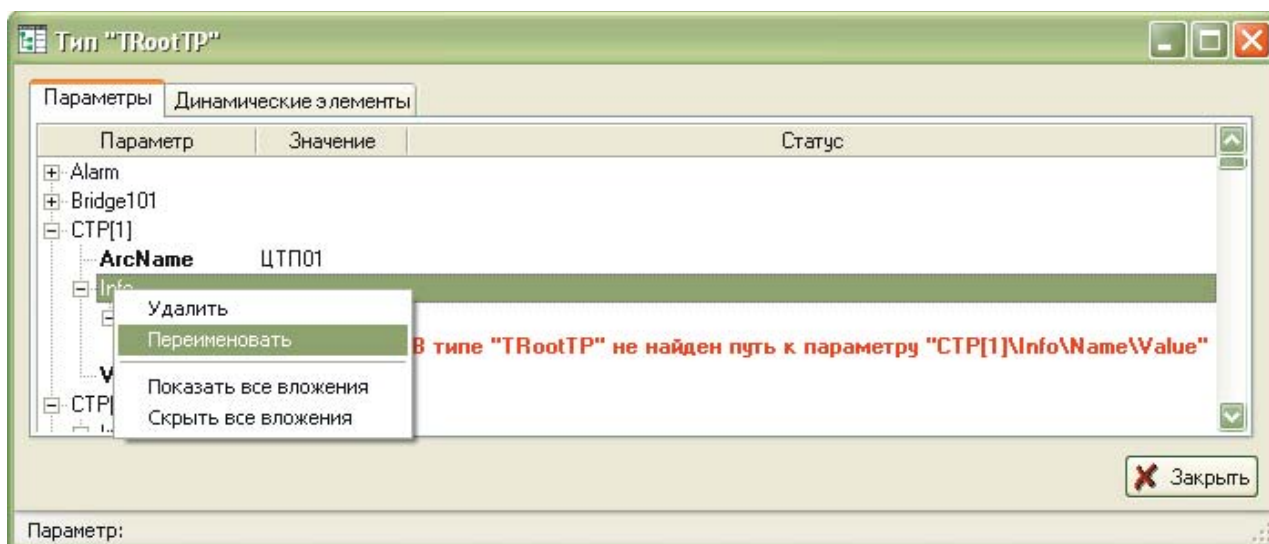


Рис. 5

Динамические элементы также как и параметры имеют иерархические имена. Также как и параметры, они могут стать некорректными при удалении или переименовании элемента, который встречается в их имяобразовании. Методы борьбы аналогичны. Можно или удалить динамический элемент, или переименовать одну из его составляющих.

9.3.5.19 Проверить тип

Чтобы проверить тип, надо или нажать на кнопку "Проверить тип"  или выбрать в главном меню пункт "Тип\Проверить тип".

При проверке типа:

- проверяется, что имя типа уникально;
- проверяется, что базовый тип известен;
- проверяется базовый тип;
- проверяется, что типы всех элементов известны;
- проверяются типы всех элементов;
- проверяются все [параметры](#) и [динамические элементы](#) типа.

9.3.6 Настройки

В программе "Конструктор OPC-модели" есть некие общие настройки. Чтобы попасть в них надо выбрать в главном меню пункт "Инструменты\Настройки".

Экран настроек содержит закладки: "Построение OPC-модели", "Переменные среды", "Параметры редактирования" и "Редактор кода".

Закладка "Построение OPC-модели" (рисунок 1) содержит опции компилятора (bcc32.exe) и линковщика (ilink32), общие для всех проектов. Некоторые из них можно дополнить для каждого конкретного проекта, используя [установки проекта](#).



Рис. 1

Закладка "Переменные среды" (рисунок 2) содержит переменные среды, которые можно использовать для задания папок и имен файлов (для этого в соответствующих местах есть кнопка )

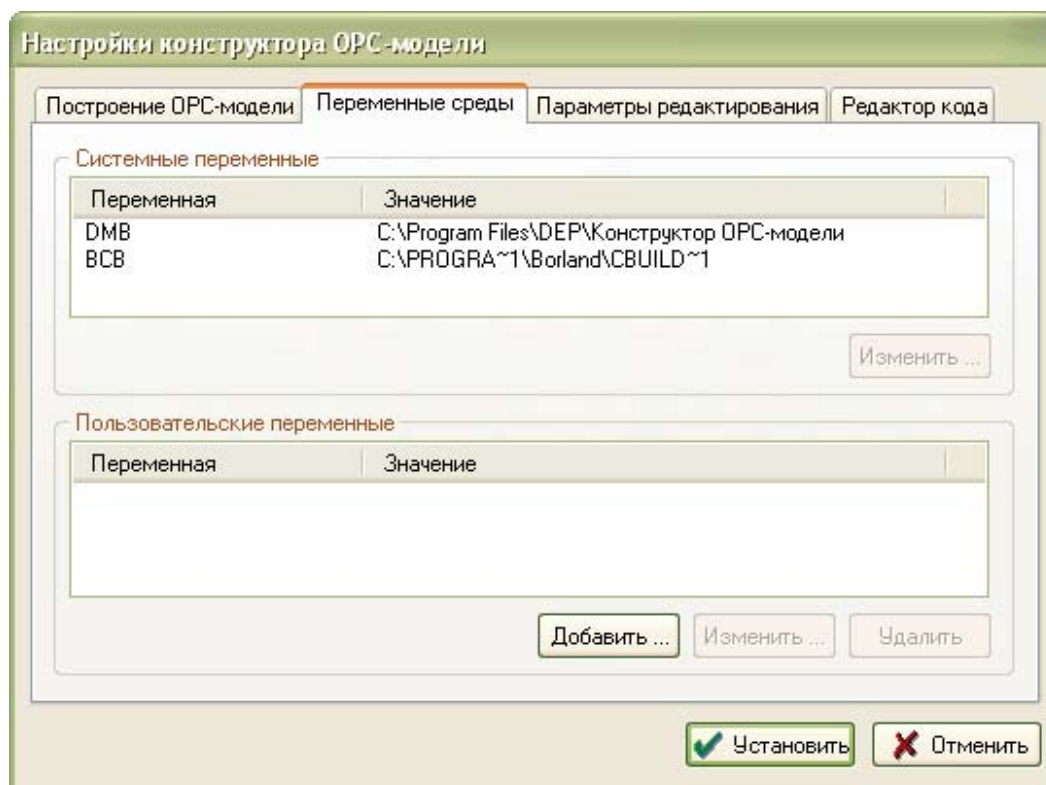


Рис. 2

Закладка "Параметры редактирования" (рисунок 3) содержит всего один параметр "не анализировать" изменения. Если эта опция установлена:

- пункты "сохранить тип", "сохранить библиотеку" доступны всегда, независимо от того, были изменения или нет;
- при сборке/построении проекта или библиотеки, файлы <библиотека.cpp> и <библиотека.h> будут формироваться независимо от того, изменился файл <библиотека.dtl> или нет.

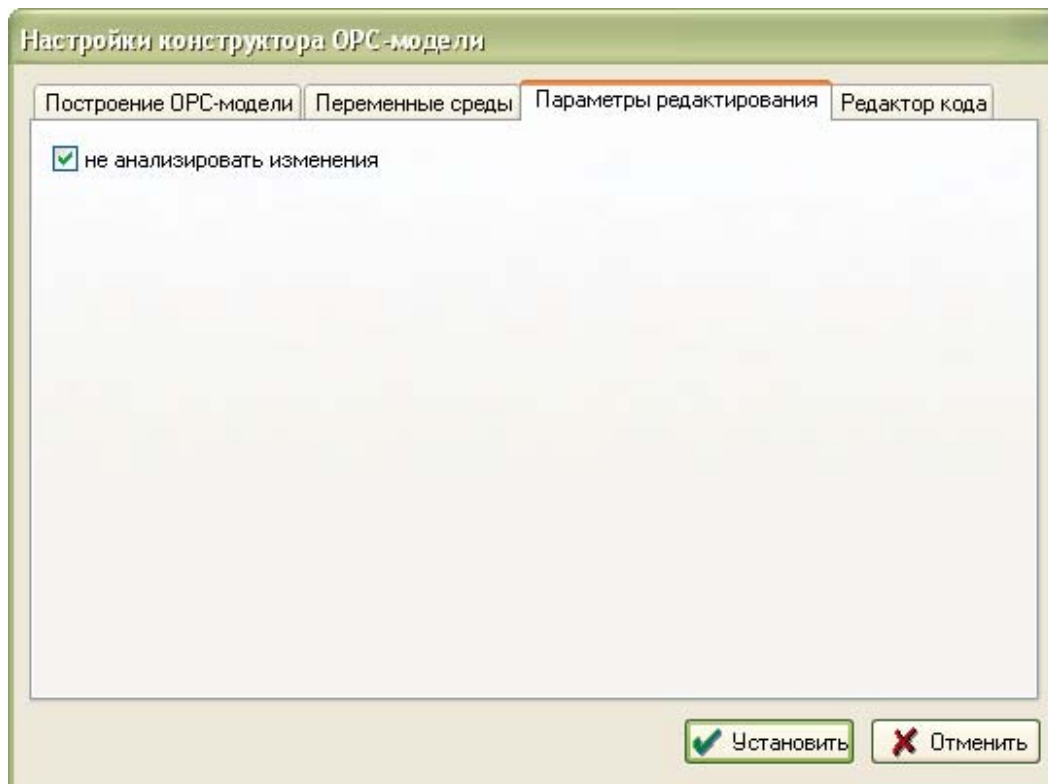


Рис. 3

Закладка "Редактор кода" (рисунок 4) содержит параметры, относящиеся к редактору функций.

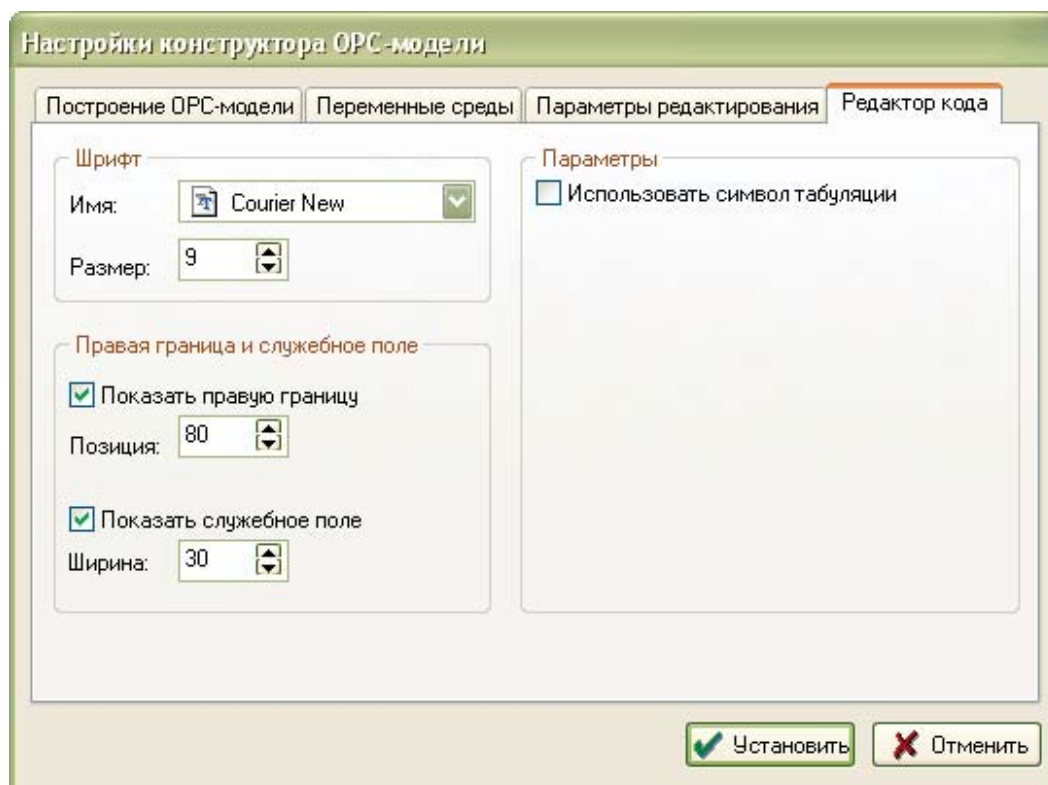


Рис. 4

9.4 Пример использования модели

9.4.1 Знакомство с Конструктором OPC-модели

9.4.1.1 Первые шаги

В основе создания OPC-модели лежат несколько очень простых понятий. Разобравшись с ними, вы быстро поймете суть создания модели в "Конструкторе OPC-модели".

Начиная создавать OPC-модель, вы должны:

1. Иметь хорошее представление о структуре технологического объекта, модель которого вы собираетесь реализовывать.
2. Уяснить функциональность, которая должна быть реализована в модели.
3. В соответствии с функциональностью определить набор интересующих вас сигналов (ТС, ТИ и ТУ).
4. Написать сценарии работы элементов (типов) модели и их взаимодействие между собой.

Не теряя времени, реализуем все эти пункты для какой-нибудь простой и, в то же время, реальной задачи. Например, пусть у нас, в некотором помещении, есть вентилятор, состояние которого мы хотим контролировать. Давайте опишем создание модели для данной задачи.

Пункт 1: структура объекта очень проста (один вентилятор) и не требует каких-либо пояснений.

- наш объект (весь объект в целом, главный тип модели).
- вентилятор (контролируемое устройство, вспомогательный тип модели).

Пункт 2: нас интересует контроль состояния вентилятора (управление мы пока не рассматриваем).

Пункт 3: в соответствии с предыдущим пунктом, нам нужен лишь один сигнал, а именно состояние вентилятора (ТС).

- вентилятор (элемент).
- состояние (ТС, свойство).

Пункт 4: наша модель не содержит никаких сценариев работы.

Пора пояснить термины, которые мы использовали при описании процесса создания модели. Модель состоит из типов, которые имеют определенный набор свойств и функций. Свойства типа это набор признаков, которыми характеризуется тип, а функции это сценарии работы типа.

Для решения задачи может потребоваться несколько типов. Тот тип, который вызывает сценарии работы других типов, называется главным. Главный тип модели присутствует всегда, других типов может быть несколько или может не быть вообще. В нашем случае у нас есть главный тип - наш объект, и один вспомогательный - вентилятор.

Теперь решим вопрос: что, в описанном выше сценарии, будете делать вы, а что среда "Конструктор OPC-модели". Дело обстоит следующим образом:


- Визуальная среда создает по вашим указаниям типы модели со всеми свойствами и формирует исходный код модели.
- Разработчик дописывает необходимые сценарии работы типов и привязывает свойства типов к интересующим его сигналам (к базам дискретов, аналогов и счетчиков).
- Визуальная среда по команде разработчика компилирует весь исходный код в результирующий модуль.

Следующий шаг: [Инструментарий "Конструктора OPC-модели"](#)

9.4.1.2 Инструментарий "Конструктора OPC-модели"

Для начала ознакомления со средой "Конструктор OPC-модели" установим программу на компьютер.

Запустите из папки с дистрибутивом программу Setup. Следуйте указаниям, которые программа-инсталлятор будет выводить на экран. По окончании работы программы Setup, на вашем компьютере будет установлена программа "Конструктор OPC-модели" в папку '...\Program Files\DEP\Конструктор OPC-модели'. В подкаталоге '...\Конструктор OPC-модели\Bin' находится файл приложения ModelBuilder.exe.

Запустите программу из меню "Пуск" - "Пуск | Программы | DeCont | Конструктор OPC-модели | Конструктор OPC-модели"  и вы попадете в визуальную среду "Конструктор OPC-модели" (рис. 1).

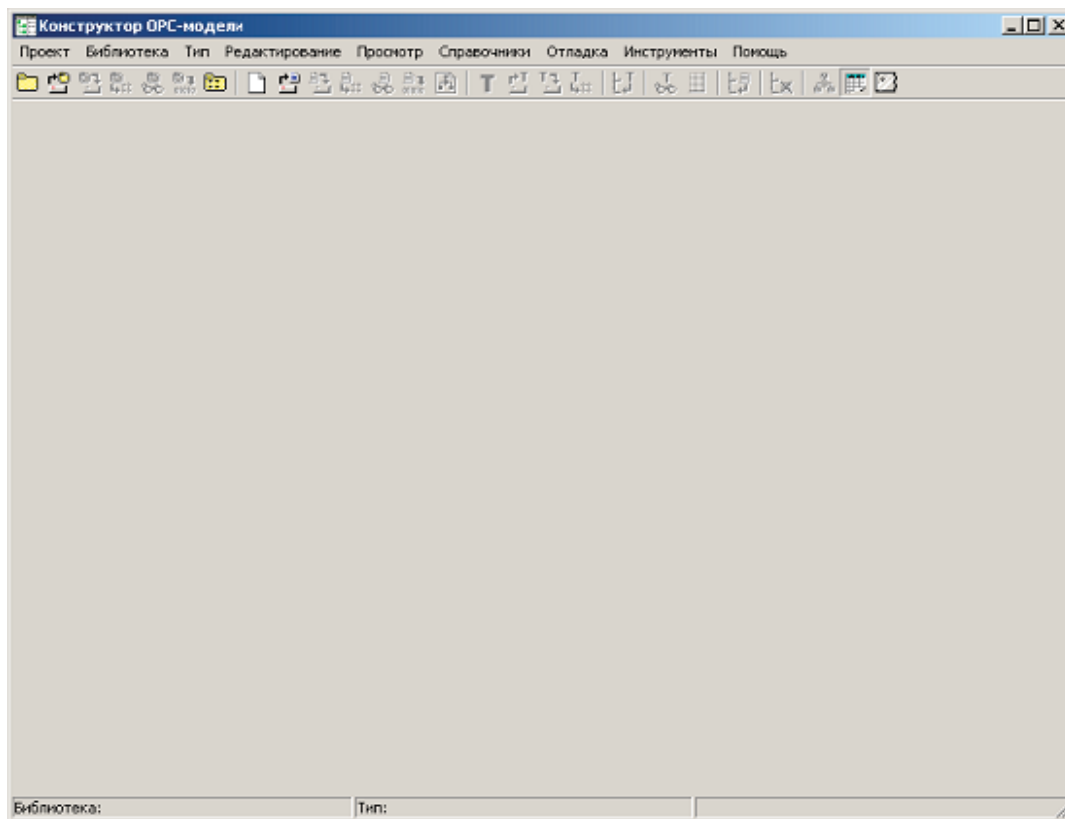


Рис. 1 Главное окно "Конструктора OPC-модели".

Рассмотрим, из каких частей состоит главное окно программы.

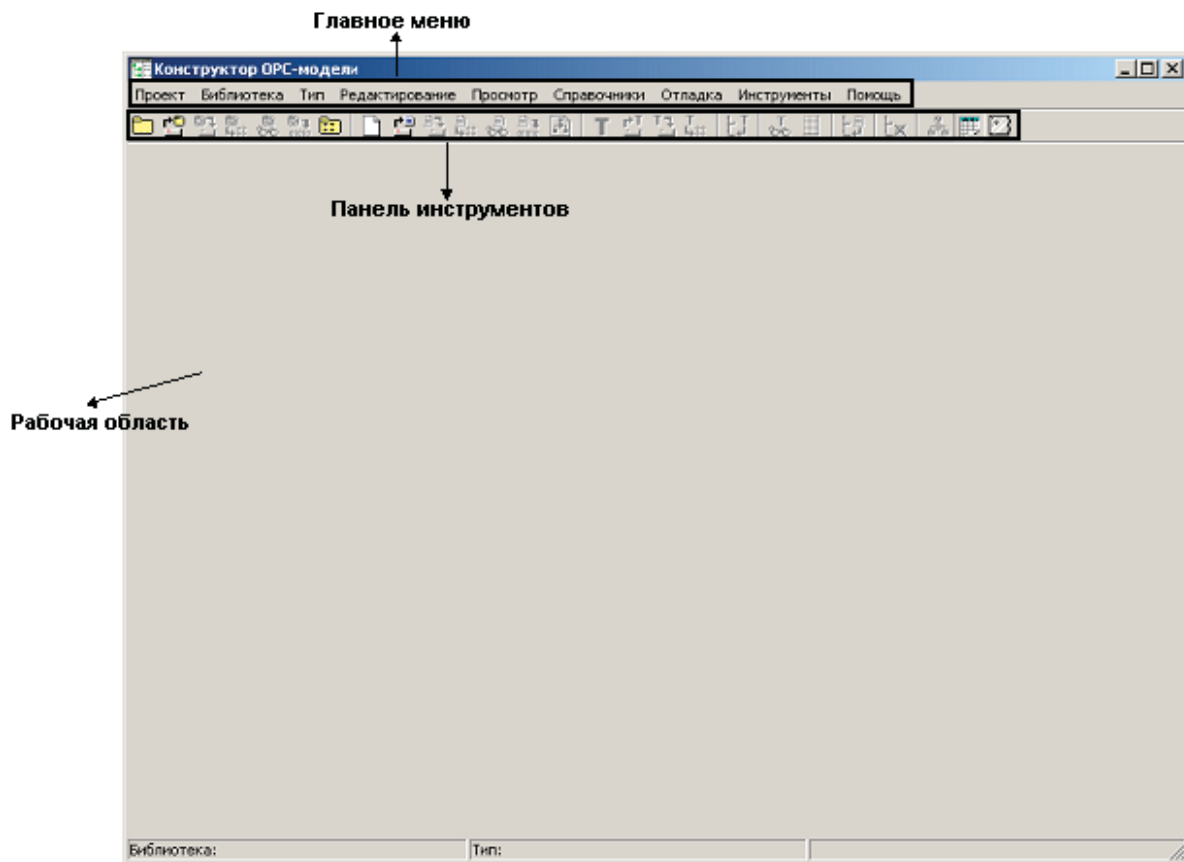


Рис.2 Составные части главного окна программы.

Обсудим кратко каждую из составных частей. Важнейшая часть - рабочая область. Пока не открыт или не создан новый проект, рабочая область пуста (рис. 1, 2). Если создать новый проект, то рабочая область примет следующий вид рис. 3.

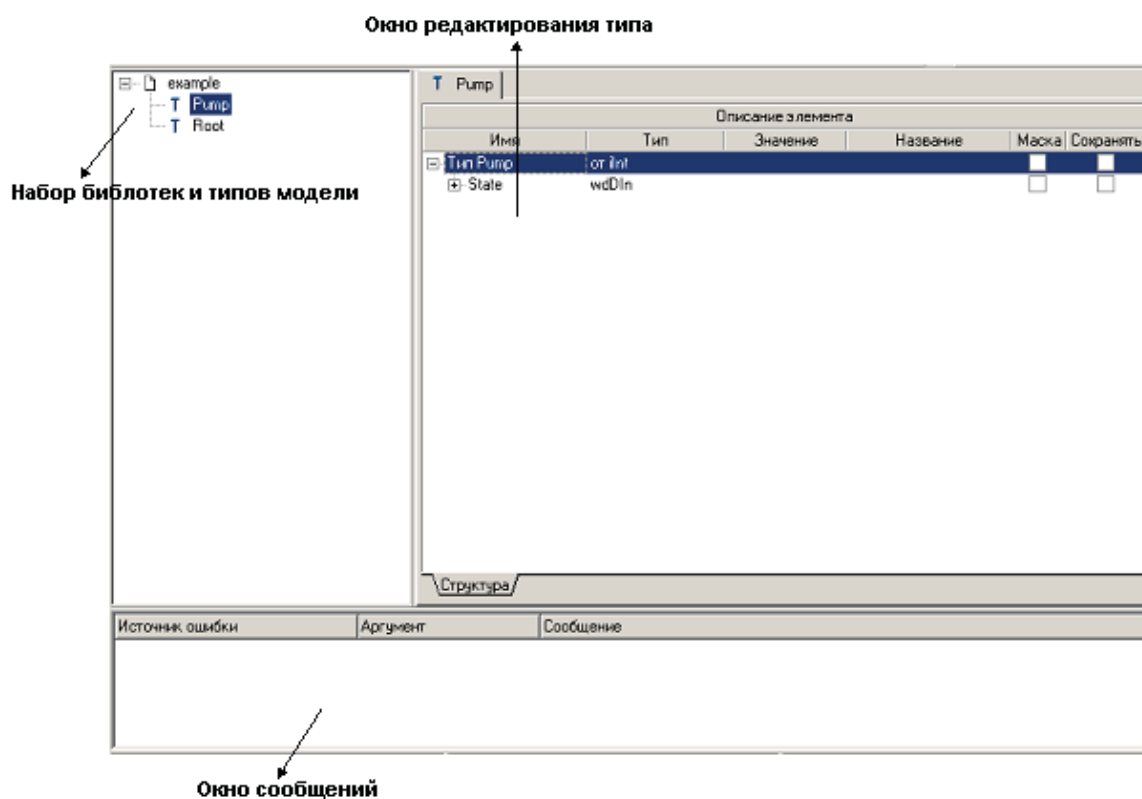


Рис. 3 Рабочая область.

Структура модели отображает набор типов, используемых в текущем проекте. Все типы разбиты по библиотекам.

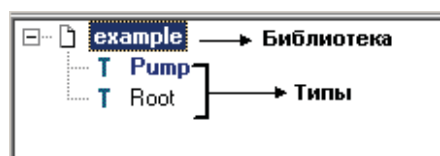


Рис. 4 Структура модели.

Окно редактирования типа отображает структуру типа (элемента). В этом окне происходит редактирование типа (добавление свойств, установка связи между свойствами и элементами баз WinDecont и т.п.).

В окне сообщений выводится различная вспомогательная информация (информация об ошибках, вывод компилятора и т.п.).

С рабочей областью все ясно, рассмотрим оставшиеся части главного окна программы. Для управления процессом создания модели в целом служит главное меню (рис. 5).

Проект Библиотека Тип Редактирование Просмотр Справочники Отладка Инструменты Помощь

Рис. 5 Главное меню.

Главное меню расположено на верху главного окна и выполняет множество служебных функций. Рассмотрим назначение

основных пунктов меню.

Меню проект служит для управления проектом (рис. 6).

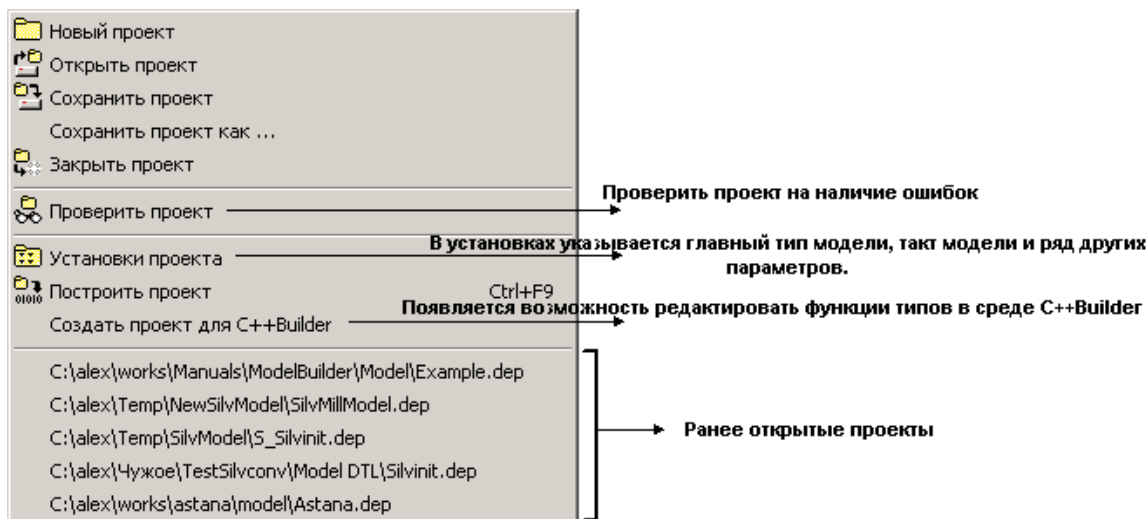


Рис. 6 Меню "Проект".

Назначение большинства пунктов понятно из их названия, для остальных приведены краткие комментарии на рисунке.

Меню библиотека служит для управления библиотеками (рис. 7).



Рис. 7 Меню "Библиотека".

Меню тип служит для управления типами (рис. 8).

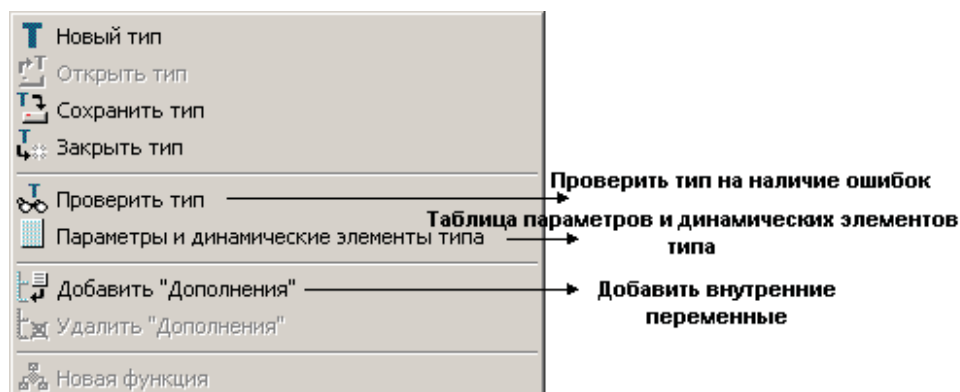


Рис. 8 Меню "Тип".

Остальные пункты меню сейчас нас интересуют меньше, поэтому мы лишь кратко опишем их предназначение.

- Редактирование - редактирование типа (вставить/удалить свойство и т.п.).
- Просмотр - просмотр содержимого типа с применением различных фильтров.
- Справочники - настройка периодов архивирования и расшифровки значений дискретов.
- Отладка - отладка процесса построения модели.
- Инструменты - используемый инструментарий (настройки компилятора, используемая версия C++ Builder).
- Помощь - помощь по программе.

Для ускорения доступа к основным операциям над проектами, библиотеками и типами служит панель инструментов (рис. 9).

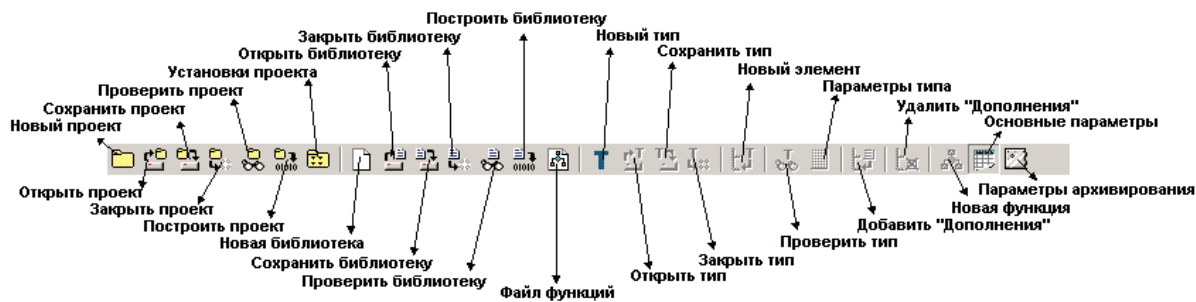


Рис. 9 Панель инструментов.


Вот вы и изучили основные элементы визуальной среды! Вы познакомились с "Конструктором OPC-модели" только в самых общих чертах, но этого достаточно, чтобы попробовать создать первую модель.

Следующий шаг: [Ваша первая модель.](#)

9.4.1.3 Ваша первая модель

Пришла пора построить вашу первую модель. Для этого построим модель объекта рассмотренного нами в [начале](#) нашего ознакомления с процессом создания модели.

Вы запустили "Конструктор OPC-модели" и видите перед собой пустую рабочую область. Начнем с создания проекта.

Переместите курсор мыши в панель инструментов и щелкните на значке  (новый проект). При этом вид рабочей области изменится.

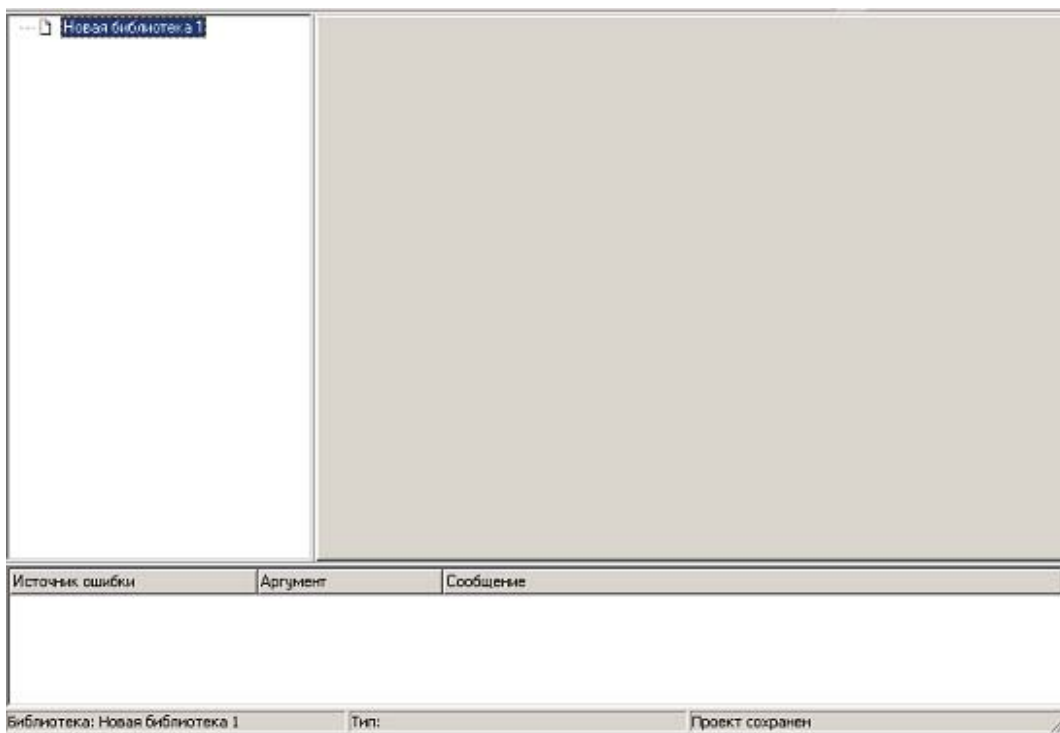



Рис. 1 Вид рабочей области после создания модели.

Появится окно структуры модели и окно сообщений. Причем, в окне структуры модели появится библиотека (новая библиотека 1). Переименовать библиотеку вы сможете в дальнейшем, когда мы будем сохранять наш проект.

Следующий наш шаг, это создание типов модели. Как вы помните, наша первая модель очень проста, и содержит всего два типа: наш объект и вентилятор. Создадим эти типы.

Сначала создадим главный тип модели - наш объект. Переместите курсор мыши в панель инструментов и щелкните на значке  (новый тип). Появится диалог (рис. 2), в котором вам будет предложено ввести название нового типа и базовый тип. Новый элемент наследует все свойства и функции базового типа.

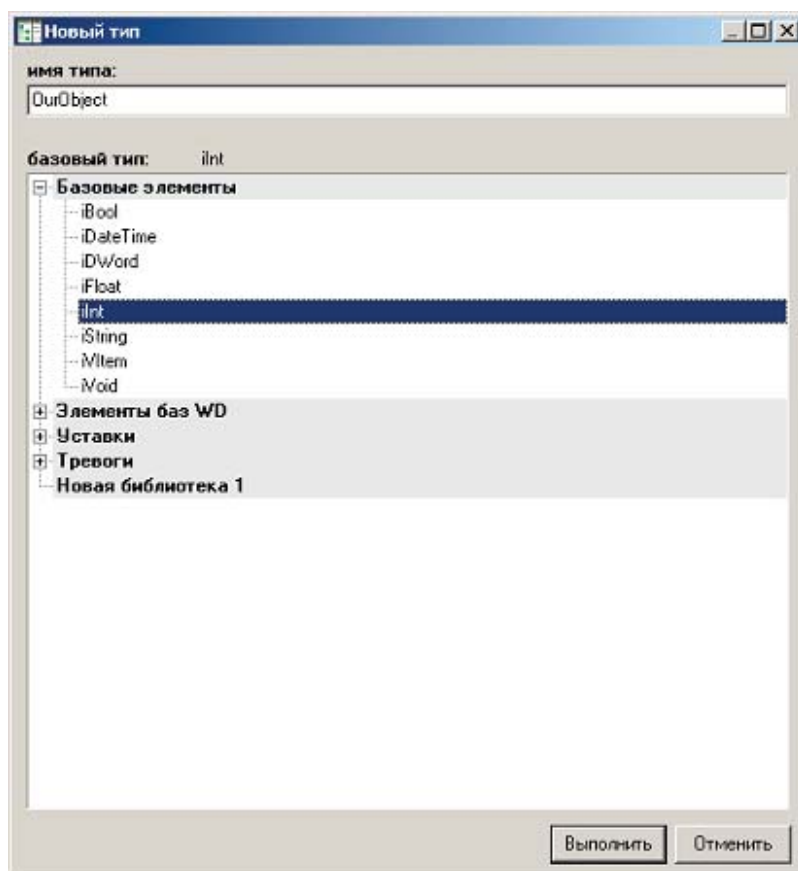


Рис. 2 Диалог создания нового типа.

!!!В названии типа можно использовать только английский язык!!!

В поле "имя типа" введите OurObject (наш объект). Далее необходимо указать базовый тип. Базовый тип можно выбирать из набора библиотек. Часть библиотек, это библиотеки по умолчанию, другие, это библиотеки, которые вы создали для вашего проекта (новая библиотека 1). В качестве базового типа выберем iInt из библиотеки базовых элементов. Этот тип характеризуется лишь одним свойством - значением, целого типа. Нажмите "Выполнить". В результате в рабочей области появится окно редактирования вновь созданного типа (рис. 3).

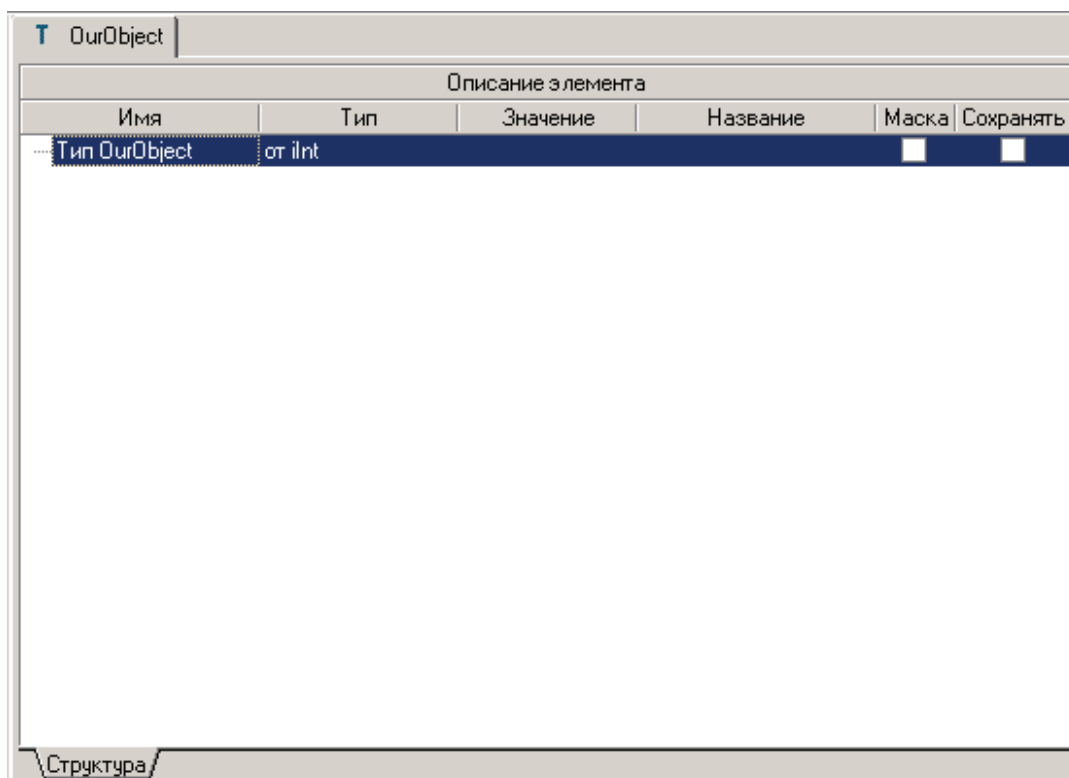


Рис. 3 Окно редактирования типа Root.


Для того чтобы новый тип добавился в библиотеку необходимо его сохранить. Щелкните мышкой на значке  (сохранить тип). После этого, тип будет добавлен в библиотеку (рис. 4).



Рис. 4 Добавление типа в библиотеку.

Создадим, теперь, наш второй тип - вентилятор (Fan) (рис. 5).

!!! Для того чтобы значок добавления типа был активен необходимо, чтобы в окне структуры модели была активна (подсвечена синим) библиотека, в которую производится добавление типа. Для этого надо щелкнуть мышкой по названию библиотеки !!!

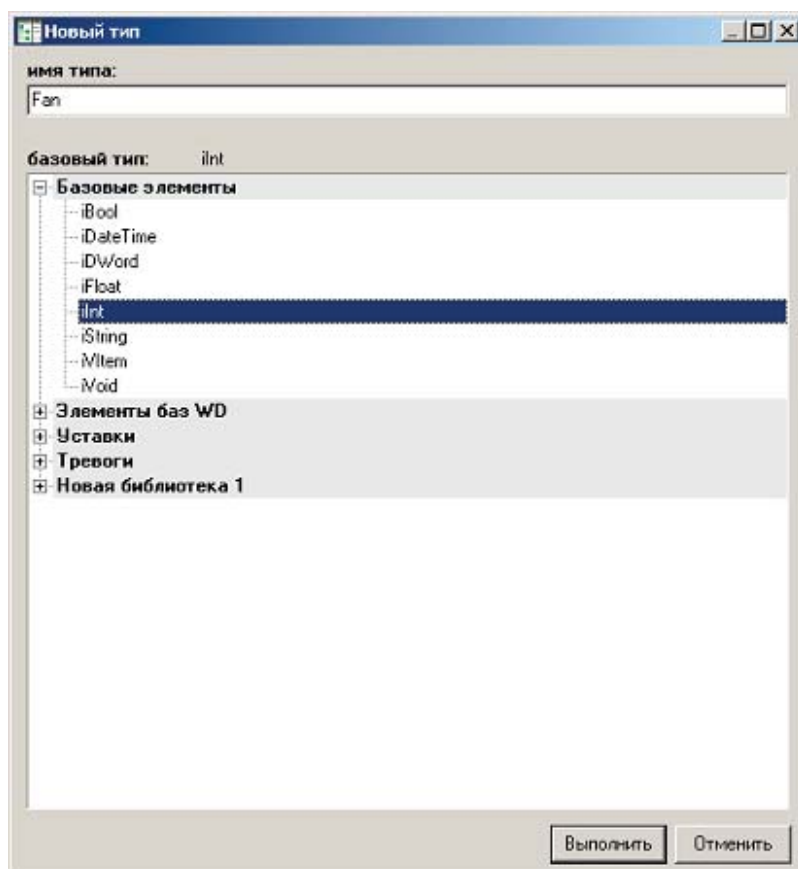


Рис. 5 Создание типа Fan.

В окне редактирования типов появится новая закладка с типом - Fan (Вентилятор).

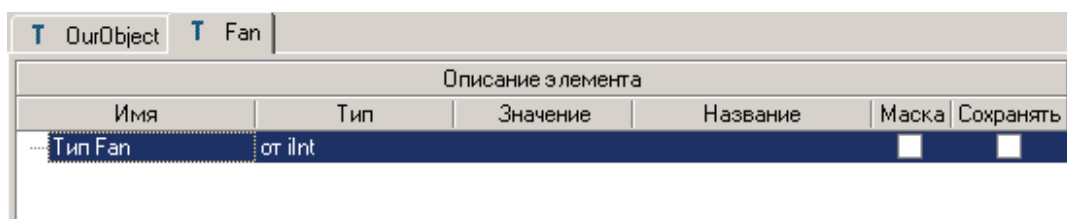



Рис. 6 Тип Fan.

У типа Fan (вентилятор), как мы помним, есть свойство - состояние. Добавим свойство "состояние" типу Fan. Для этого щелкните мышкой на значке  (новый элемент). Появится диалог, в котором вам будет предложено ввести название свойства и его тип (рис. 7).

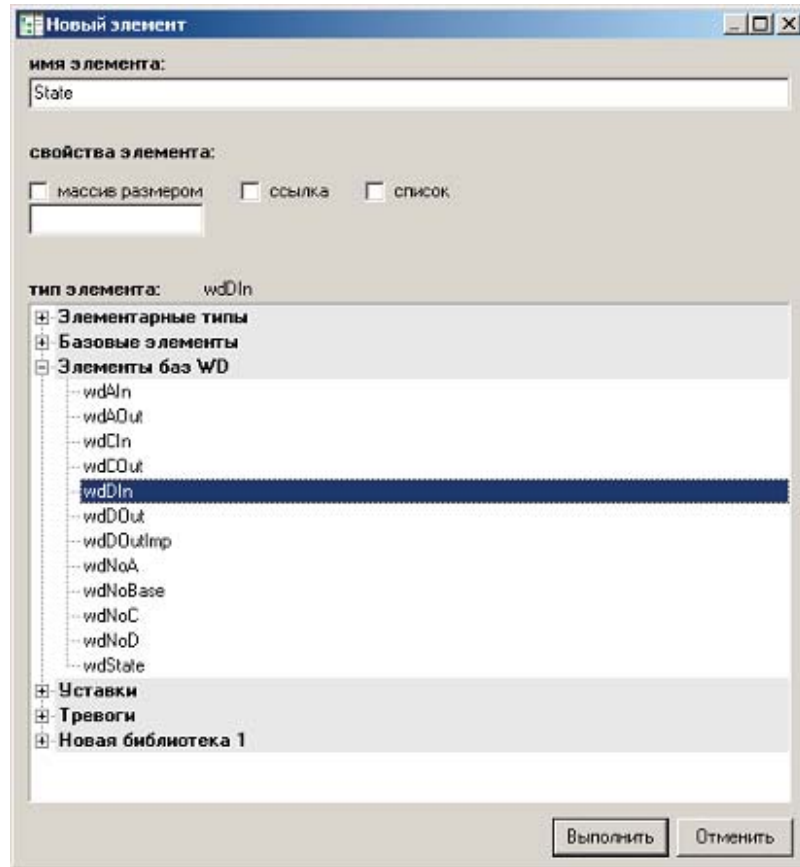


Рис. 7 Добавление нового свойства.

В поле "имя типа" впишите - State (состояние). Далее надо определиться с типом свойства. Тип свойства можно выбирать из того же набора библиотек, что и для базового типа. В нашем случае свойство "состояние" является сигналом от датчика, контролирующего состояние вентилятора. Значит, мы должны выбрать тип свойства из библиотеки "Элементов баз WD". Сигнал от датчика у нас будет дискретным, поэтому выберем тип - wdDIn (чтение дискрета). Нажмите кнопку "Выполнить". В результате на экране редактирования типа Fan появится новое свойство - State (рис. 8).

Описание элемента					
Имя	Тип	Значение	Название	Маска	Сохранять
Тип Fan	ot ilnt			<input type="checkbox"/>	<input type="checkbox"/>
State	wdDIn			<input type="checkbox"/>	<input type="checkbox"/>
No	wdNoD	0			
Inv	bool	False			
Type	int	0			

Рис. 8 Свойство State типа Fan.

Так как все названия типов и свойств вводятся только на английском языке, может возникнуть некоторая путаница. Поэтому, давайте в поле "Название" свойства State введем расшифровку названия свойства на русском языке. Для этого наведите курсор мыши на строку, содержащую свойство, и щелкните мышкой. Вся строка, при этом, будет выделена синим цветом.

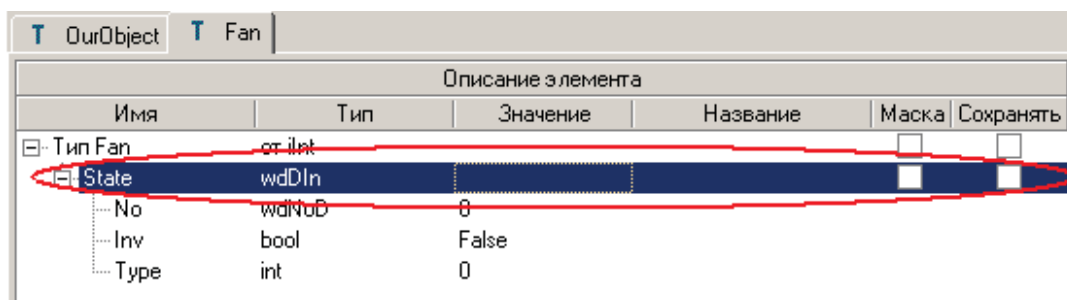


Рис. 9 Выделение строки для редактирования.

После этого щелкните мышкой в ячейке "Название". При этом вокруг ячейки появится пунктирная рамка. Теперь, введите с клавиатуры название - "состояние". Нажмите Enter. Все, теперь свойство State имеет пояснение.

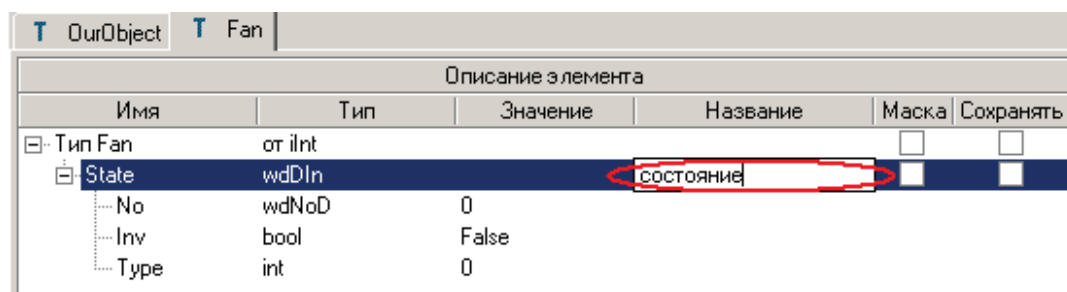



Рис. 10 Редактирование названия свойства.

Сохраните тип Fan. Для этого щелкните мышкой по иконке . После этого, в нашей библиотеке появится новый тип Fan.



Рис. 11 Добавление типа Fan в библиотеку.

Замечательно! Мы создали два наших типа, что же нам делать с ними теперь? Всё очень просто, мы должны добавить типу OurObject (наш объект) свойство "вентилятор_1" (Fan_1). И после этого привязать состояние вентилятора к базе WinDecont. Давайте сделаем все это по порядку. Сначала добавим свойство "вентилятор_1" типу OurObject. Для этого

активируйте закладку - OurObject. Щелкните мышкой на значке , введите название свойства - Fan_1 и укажите тип - Fan (из библиотеки - Новая библиотека 1).

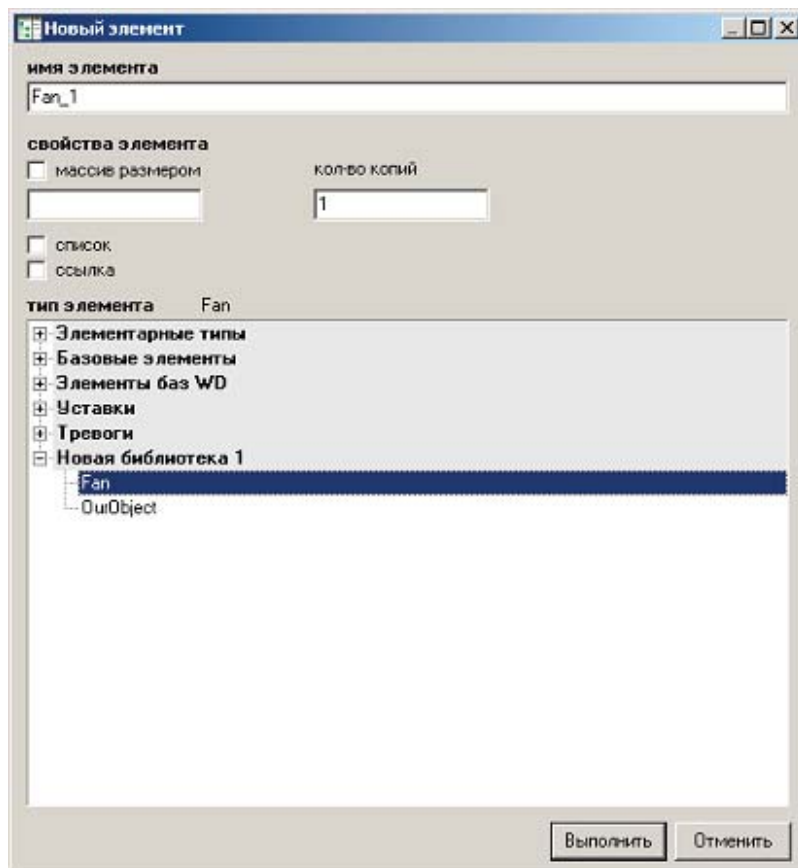


Рис. 12 Добавление свойства Fan_1 типу OurObject.

После этого в окне редактирования типа OurObject появится свойство Fan_1.

Описание элемента					
Имя	Тип	Значение	Название	Маска	Сохранять
Тип OurObject	от int			<input type="checkbox"/>	<input type="checkbox"/>
Fan_1	Fan			<input type="checkbox"/>	<input type="checkbox"/>
State	wdIn		состояние	<input type="checkbox"/>	<input type="checkbox"/>
No	wdNoD	0			
Inv	bool	False			
Type	int	0			

Рис. 13 Свойство Fan_1 типа OurObject.

Для того, чтобы свойство State вентилятора Fan_1 отражало реальное состояние вентилятора, нам необходимо привязать его к базе дискретов WinDecont. Для этого выделите мышкой ячейку "Значение" строки No и впишите 1.

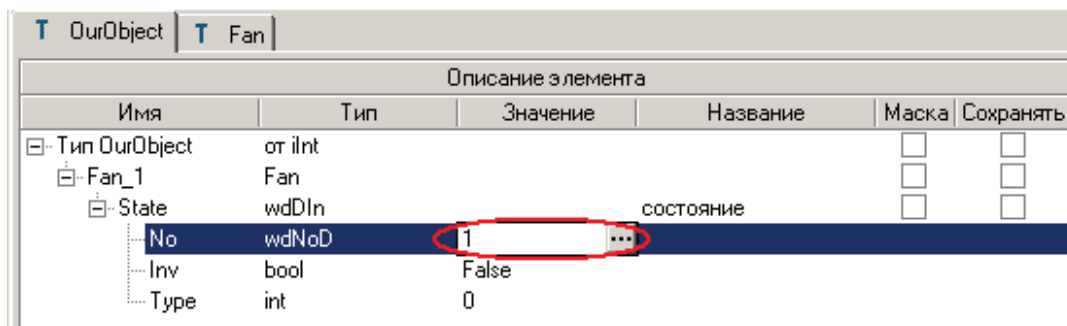


Рис. 14 Привязывание свойства State к базе дискретов WinDecont.

Тем самым мы говорим, что значение свойства State для вентилятора Fan_1 берется из первого дискрета в базе параметров WinDecont.

Давайте, также, впишем название для вентилятора Fan_1.

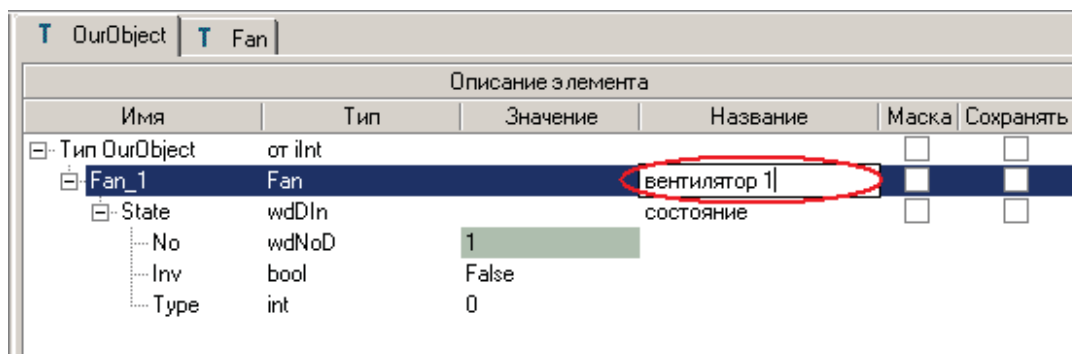



Рис. 15 Редактирование названия свойства Fan_1.

Все! Редактирование типов закончено. Давайте сохраним результат наших трудов. Для этого щелкните мышкой по иконке  (сохранить проект). Сначала "Конструктор OPC-модели" предложит ввести имя для библиотеки, а потом и для всего проекта. Библиотеку назовем FirstLib, а проект - FirstModel.

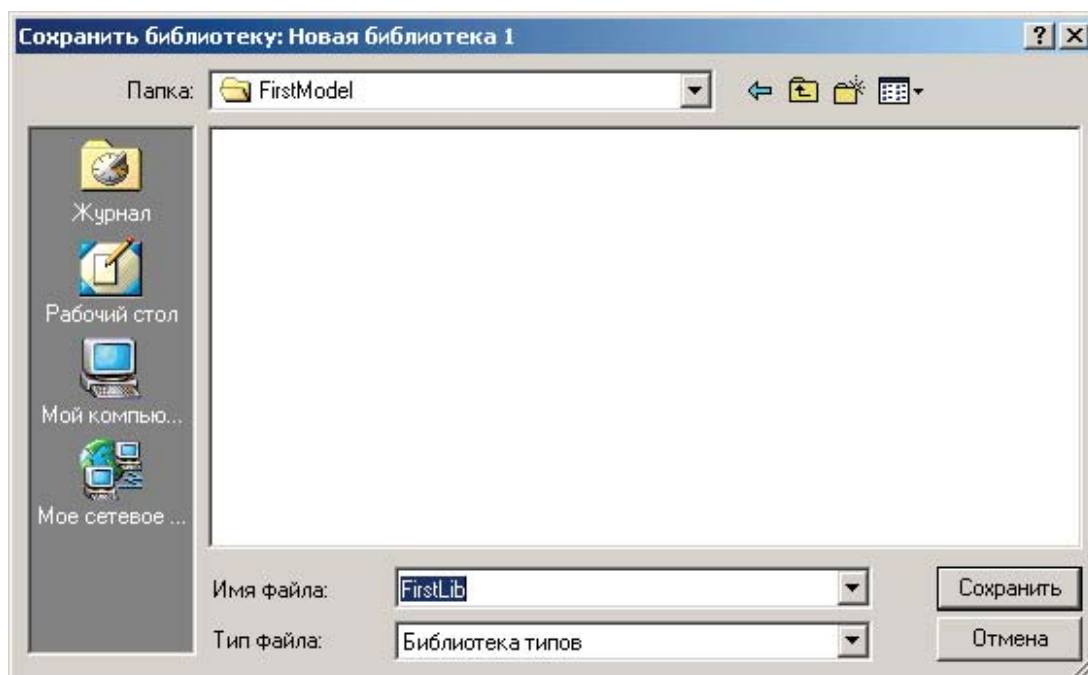


Рис. 16 Диалог сохранения библиотеки.

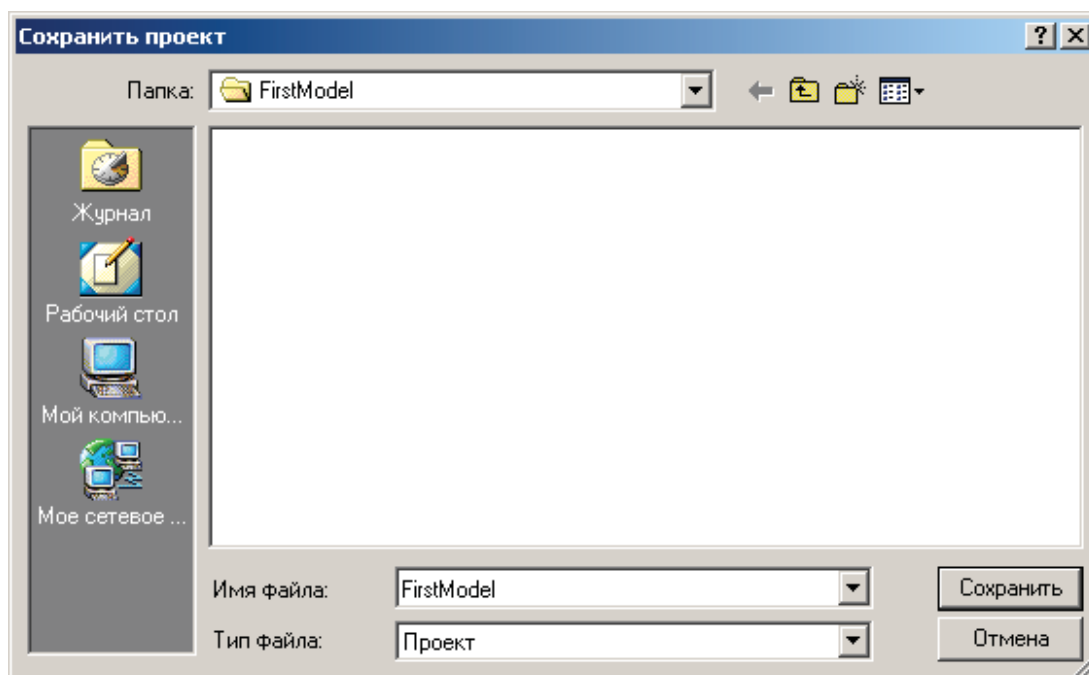



Рис. 17 Диалог сохранения проекта.

Все файлы, которые относятся к проекту, будут сохранены по заданному пути. Сохраненный проект может быть открыт для доработки в любой момент.

Нам осталось установить свойства проекта и собрать проект. Для того чтобы установить свойства проекта щелкните

мышкой по иконке  (установки проекта). В открывшемся диалоге вам надо указать главный тип модели и такт выполнения модели. Главный тип нашей модели, как вы помните это OurObject, а так установите равным 100 мс.

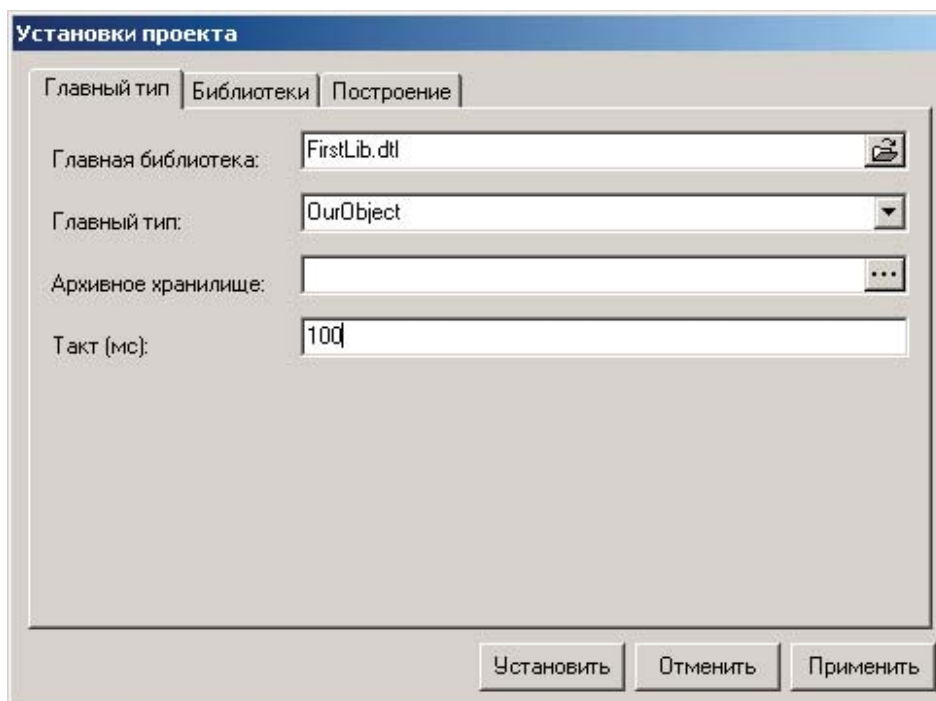



Рис. 18 Установки проекта.

Постройте проект, для этого щелкните мышкой по иконке  (построить проект). После этого в окне сообщений появится вывод компилятора. Давайте посмотрим, что же нам написал компилятор. С удивлением, вы обнаруживаете ошибку, компилятор пишет, что он не знает такой структуры - Fan.

```
14:16:33: Генерация кода C:\alex\works\Manuals\ModelBuilder\Models\FirstModel\RESULT\FirstLib.h
14:16:33: Генерация кода C:\alex\works\Manuals\ModelBuilder\Models\FirstModel\RESULT\FirstLib.cpp
14:16:33: Генерация кода C:\alex\works\Manuals\ModelBuilder\Models\FirstModel\RESULT>Main.h
14:16:33: Генерация кода C:\alex\works\Manuals\ModelBuilder\Models\FirstModel\RESULT>Main.cpp
14:16:33: Построение MAKE-файла C:\alex\works\Manuals\MODELB-1\Models\FIRSTM-1\RESULT\tempcap.nak
14:16:33: Выполнение MAKE-файла ...
MAKE Version 5.2 Copyright (c) 1987, 2000 Borland
D:\PROGRA-1\Borland\CBUILD-2\BIN\bcc32.exe -MD -O2 -b -k- -vi -H=dal.csn -Vx -Ve -X- -a8 -tWD -tWH -c -D_DEBUG
Borland C++ 5.6 for Win32 Copyright (c) 1993, 2002 Borland
C:\alex\works\Manuals\MODELB-1\Models\FIRSTM-1\RESULT\FirstLib.cpp:
Error E2450 C:\alex\works\Manuals\MODELB-1\Models\FIRSTM-1\RESULT\FirstLib.h 17: Undefined structure 'Fan'
Error E2449 C:\alex\works\Manuals\MODELB-1\Models\FIRSTM-1\RESULT\FirstLib.h 17: Size of 'Fan_1' is unknown or zero
Error E2450 C:\alex\works\Manuals\MODELB-1\Models\FIRSTM-1\RESULT\FirstLib.h 17: Undefined structure 'Fan'
*** 3 errors in Compile ***
C:\alex\works\Manuals\MODELB-1\Models\FIRSTM-1\RESULT>Main.cpp:
Loaded pre-compiled headers.
Error E2450 C:\alex\works\Manuals\MODELB-1\Models\FIRSTM-1\RESULT\FirstLib.h 17: Undefined structure 'Fan'
Error E2449 C:\alex\works\Manuals\MODELB-1\Models\FIRSTM-1\RESULT\FirstLib.h 17: Size of 'Fan_1' is unknown or zero
Error E2450 C:\alex\works\Manuals\MODELB-1\Models\FIRSTM-1\RESULT\FirstLib.h 17: Undefined structure 'Fan'
*** 3 errors in Compile ***

** error 1 ** deleting C:\alex\works\Manuals\MODELB-1\Models\FIRSTM-1\RESULT>Main.obj
```

Рис. 19 Вывод компилятора с сообщением об ошибке.

Что же мы сделали не правильно?

!!!В дереве модели типы, которые используются в других типах, должны быть расположены выше последних !!!

Исправим ситуацию и поместим тип Fan перед типом OurObject. Щелкните по типу Fan, в окне структуры модели, правой кнопкой мыши и, удерживая ее, переместите тип Fan перед OurObject.



Рис. 20 Правильная структура модели.

Сохраните изменения () и заново постройте модель ().

```

14:29:11: Генерация кода C:\alex\works\Manuals\ModelBuilder\Models\FirstModel\RESULT\FirstLib.h
14:29:11: Генерация кода C:\alex\works\Manuals\ModelBuilder\Models\FirstModel\RESULT\FirstLib.cpp
14:29:11: Генерация кода C:\alex\works\Manuals\ModelBuilder\Models\FirstModel\RESULT>Main.h
14:29:11: Генерация кода C:\alex\works\Manuals\ModelBuilder\Models\FirstModel\RESULT>Main.cpp
14:29:11: Построение MAKE-файла C:\alex\works\Manuals\MODELB-1\Models\FIRSTM-1\RESULT\tempcar.mak
14:29:11: Выполнение MAKE-файла ...
MAKE Version 5.2 Copyright (c) 1987, 2000 Borland
D:\PROGRAM-1\Borland\CBUILD-2\BIN\bcc32.exe -UD -O2 -b -k- -vi -H=dml.csm -Vx -We -X- -a8 -tWD -tWH -c -D_DEBUG
Borland C++ 5.6 for Win32 Copyright (c) 1993, 2002 Borland
C:\alex\works\Manuals\MODELB-1\Models\FIRSTM-1\RESULT\FirstLib.cpp:
C:\alex\works\Manuals\MODELB-1\Models\FIRSTM-1\RESULT>Main.cpp:
Loaded pre-compiled headers.

14:29:15: Выполнение MAKE-файла завершено
14:29:15: Построение LINK-файла
14:29:15: Сборка проекта ...
Turbo Incremental Link 5.60 Copyright (c) 1997-2002 Borland

14:29:17: Сборка проекта завершена

```

Рис. 21 Вывод компилятора без сообщений об ошибках.

Результаты работы компилятора лежат в папке ...путь к вашему проекту\RESULT. Среди этих файлов нас больше всего интересует Model.dll. Это и есть ваша модель, которую мы будем запускать в WinDecont'e.

Можете себя поздравить! Ваша первая модель готова.

Следующий шаг: [Запуск модели.](#)

9.4.1.4 Запуск модели

После того, как вы построили модель ее можно запустить на исполнение. Это делается в программе WinDecont. Запустите WinDecont и откройте окно с параметрами (смотри помощь по WinDecont) (рис. 1).

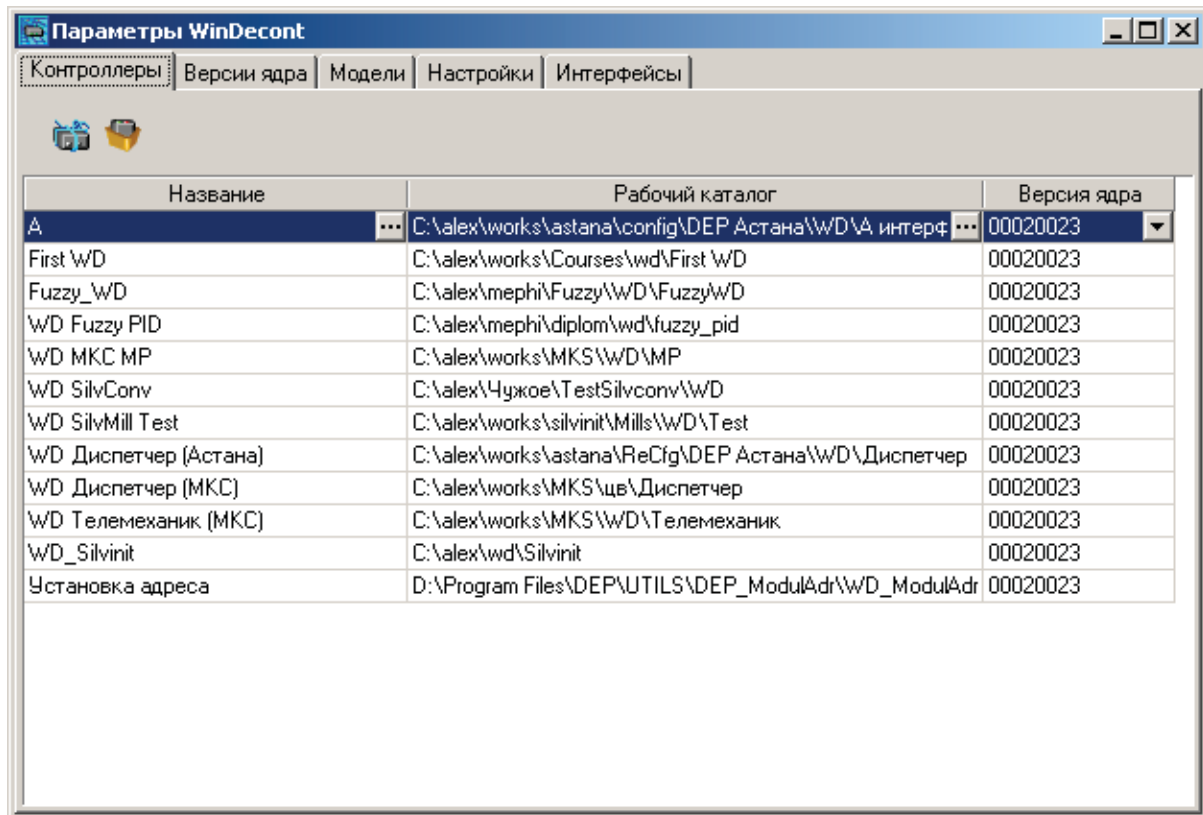



Рис. 1 Параметры WinDecont.

Перейдите в закладку "Модели" и щелкните мышкой по значку создания новой модели . В диалоге укажите название модели - FirstModel и путь к файлу Model.dll - ... путь к вашей модели\RESULT.

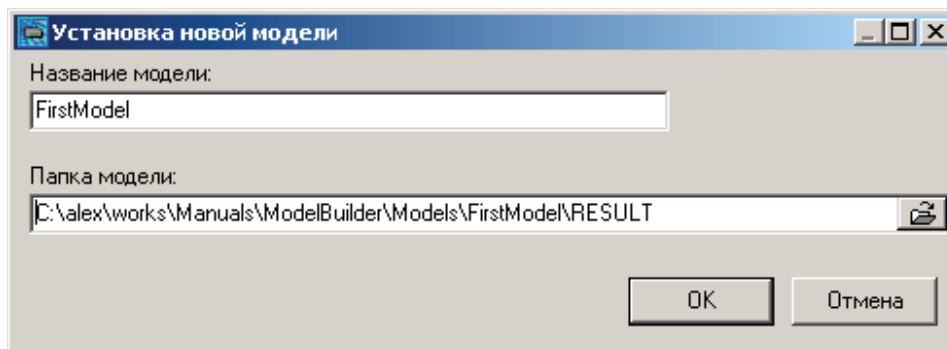


Рис. 2 Установка новой модели.

После этого, новая модель появится в списке установленных моделей.

Название	Рабочий каталог
Astana	C:\alex\works\astana\model\RESULT
Astana twin	C:\alex\works\Astana twin\Model
CompModel	C:\Temp\CompModel\RESULT
FuzzyPID	C:\alex\mephi\diplom\models\FuzzyPIDmodel\RESULT
Silvinit	C:\alex\DepPri\CKPЧЗ-Обогащение\S_Model\RESULT
SilvMill	C:\alex\Temp\NewSilvModel\RESULT
TempSilvModel	C:\alex\Temp\SilvModel
Test Astana	C:\alex\works\astana\OUTGOING\2003-02-07\data\Модель
TestFan	C:\Temp\TestFan\RESULT
TestSilv	C:\alex\Чужое\TestSilvconv\Model DTL\RESULT
Traffic	C:\alex\mephi\diplom\RESULT
FirstModel	C:\alex\works\Manuals\ModelBuilder\Models\FirstModel\RESULT

Рис. 3 Список установленных моделей.

Закройте окно параметров WinDecont. В поле модель из выпадающего списка выберите FirstModel.

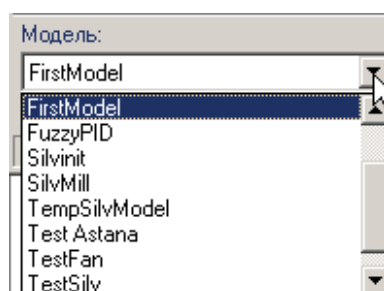


Рис. 4 Выбор модели.

Щелкните мышкой по значку запуска модели . Модель запущена!

Следующий шаг: [Итоги.](#)

9.4.1.5 Итоги

Пора подвести первые итоги:

- Вы знаете с чего начать разработку модели
- Изучили почти всю необходимую терминологию.
- Представляете свою как роль, так и роль "Конструктора OPC-модели" в процессе построения модели
- Познакомились с визуальной средой "Конструктор OPC-модели"
- Построили свою первую модель

Вы добились значительных успехов и, конечно, полны решимости приступить к построению более сложных моделей. Только, перед этим, мы должны выяснить: для чего же нам нужна модель? Ответ прост: модель нам нужна для построения автоматического рабочего места (АРМ) диспетчера и/или телемеханика некоторого объекта. АРМ диспетчера: выводит на экран компьютера графическую информацию о состоянии объекта и его параметрах, дает возможность производить управление объектом и архивировать любые параметры и управления объекта. В такой задаче модель является промежуточным звеном между объектом и его графическим представлением. Поэтому, чтобы закончить нашу первую задачу, дополним модель графическим отображением состояния вентилятора. Для этого нам надо узнать, каким же образом строится графическое отображение.

Следующий шаг: [Знакомство с "OPC-дизайнером"](#) .

9.4.2 Знакомство с OPC-дизайнером

9.4.2.1 Введение

Создание отображения происходит в среде C++Builder с использованием специального объекта `depOPCDesigner`. В этом описании мы не будем подробно останавливаться на особенностях работы в среде C++Builder, поэтому, прежде чем приступить к построению отображения, надо освежить свои знания по визуальной среде C++Builder и иметь минимальные знания по синтаксису C++.

Я уверен, что вам уже не терпится создать отображение для нашей модели. Давайте начнем.

Следующий шаг: [Первые шаги](#).

9.4.2.2 Первые шаги

Процесс создания отображения можно разбить на несколько этапов:

1. Нарисовать на бумаге внешний вид отображения и определится, какие элементы (кнопки, текст, картинки) будут использоваться в отображении.
2. Создать новый проект в C++Builder и нарисовать отображение на форме проекта с помощью визуальных компонентов.
3. Определится, какие свойства визуальных компонентов мы хотим изменять (цвет фона, текст, ...). Привязать эти свойства к элементам модели.
4. Собрать и запустить проект.

Смысл всех этапов понятен, не очевидным остается одно: что, значит, привязать свойства к элементам модели и как это сделать. Давайте это выясним. Привязать свойство, значит, установить взаимосвязь между состоянием элемента модели и свойством объекта отображения. Например, изменять цвет объекта в зависимости от значения элемента модели. Связывание свойств осуществляется при помощи специального объекта - `depOPCDesigner`. Поэтому, перед тем, как приступить к созданию нашего первого отображения, мы должны познакомиться с компонентом `depOPCDesigner`.

Следующий шаг: [Инструментарий "OPC-дизайнера"](#) .

9.4.2.3 Инструментарий "ОПС-дизайнера"

Я уверен, что C++Builder уже установлен на вашем компьютере (если нет, то перед тем как двигаться дальше, вы должны установить C++Builder, версии не ниже 6, на ваш компьютер). Теперь установим "ОПС-дизайнер".

Запустите из папки с дистрибутивом компонентов для C++Builder программу Setup. Следуйте указаниям, которые программа-инсталлятор будет выводить на экран. По окончании работы программы Setup, на вашем компьютере будут установлены компоненты для C++Builder фирмы DEP. Появятся две новых закладки, в палитре компонентов, DEP и DEP OPC. Закладка DEP OPC содержит "ОПС-дизайнер" (depOPCDesigner) и depOPCPanel. Закладка DEP содержит ряд прикладных компонентов.

Запустите C++Builder и откройте, в палитре компонентов, закладку DEP OPC.

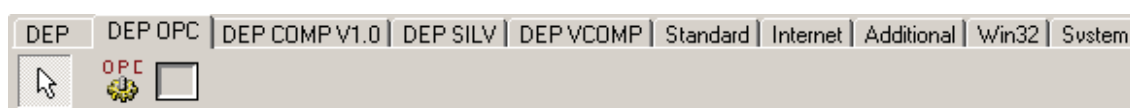


Рис. 1 Закладка компонентов DEP OPC.

Закладка DEP OPC содержит два компонента - depOPCDesigner и depOPCPanel.

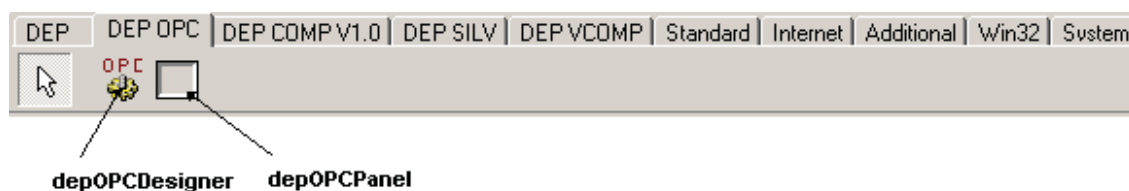


Рис. 2 Компоненты depOPCDesigner и depOPCPanel.

Переместите курсор мыши в палитру компонентов и щелкните на значке depOPCDesigner . Затем переместите курсор в любое место формы и щелкните еще раз. Объект depOPCDesigner окажется на форме (рис. 3).

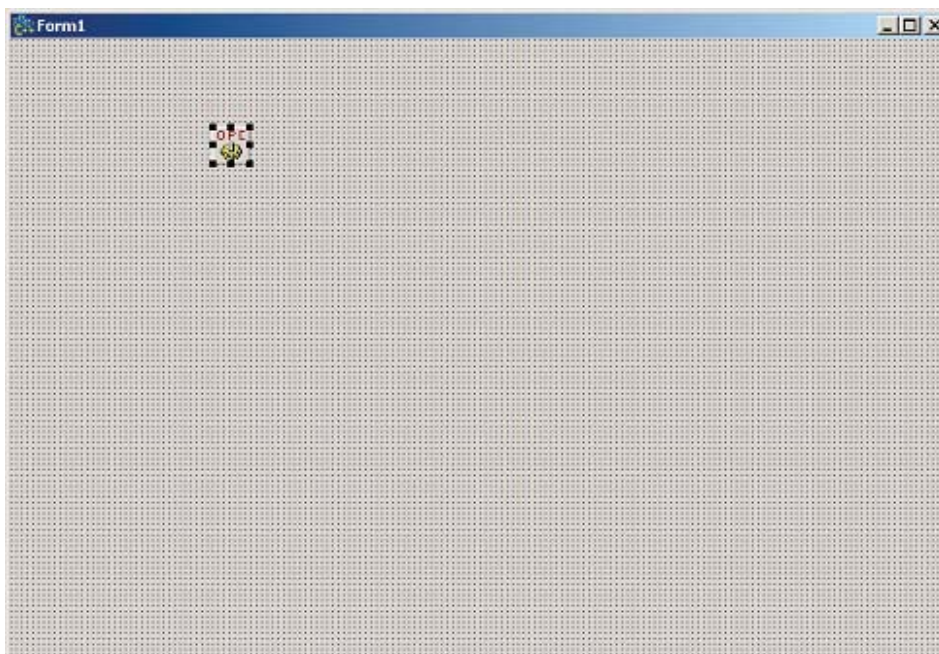


Рис. 3 Объект depOPCDesigner на форме.

Объект `depOPCDesigner` не визуальный, поэтому его расположение на форме не имеет значения. Каким же образом работает `depOPCDesigner`? Если кратко, то он связывает свойства компонентов C++Builder и элементов модели, таким образом, при изменении свойства элемента модели, происходит изменение соответствующего свойства программного компонента. Рассмотрим, как происходит привязка свойств и как она работает.

Запустите нашу модель. Положите на форму объект `Shape` (закладка `Additional`).

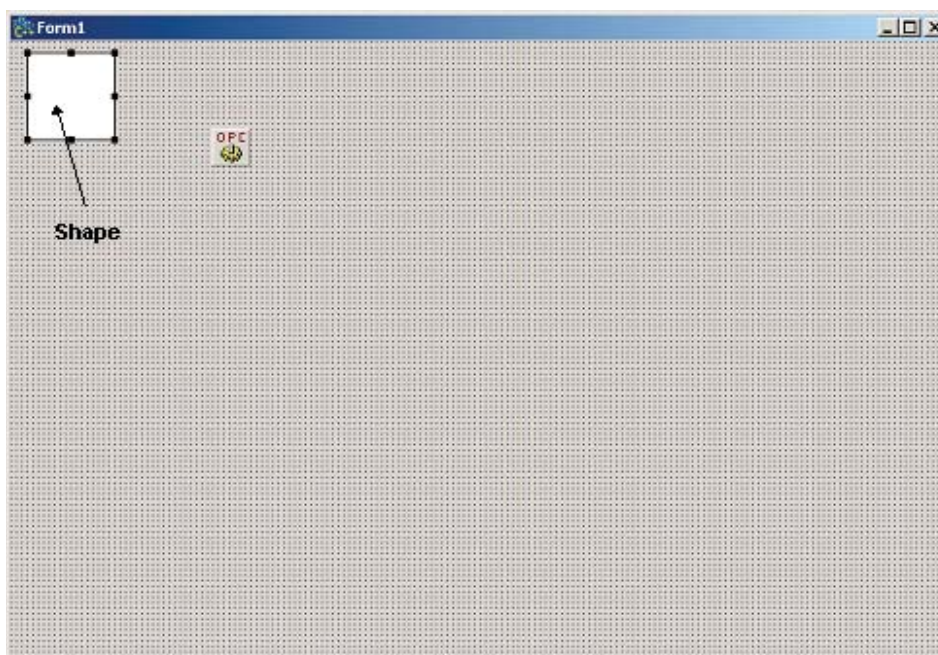


Рис. 4 Объект Shape на форме.

Щелкните правой кнопкой мышки по объекту Shape и выберите из выпадающего меню "Редактор OPC состояний".

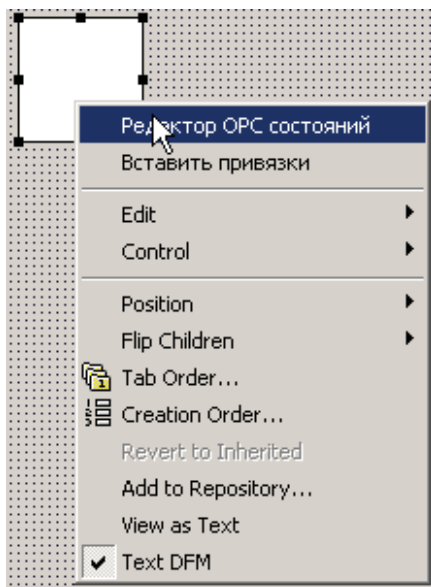


Рис. 5 Выпадающее меню.

При этом откроется редактор OPC состояний (OPC-дизайнер).

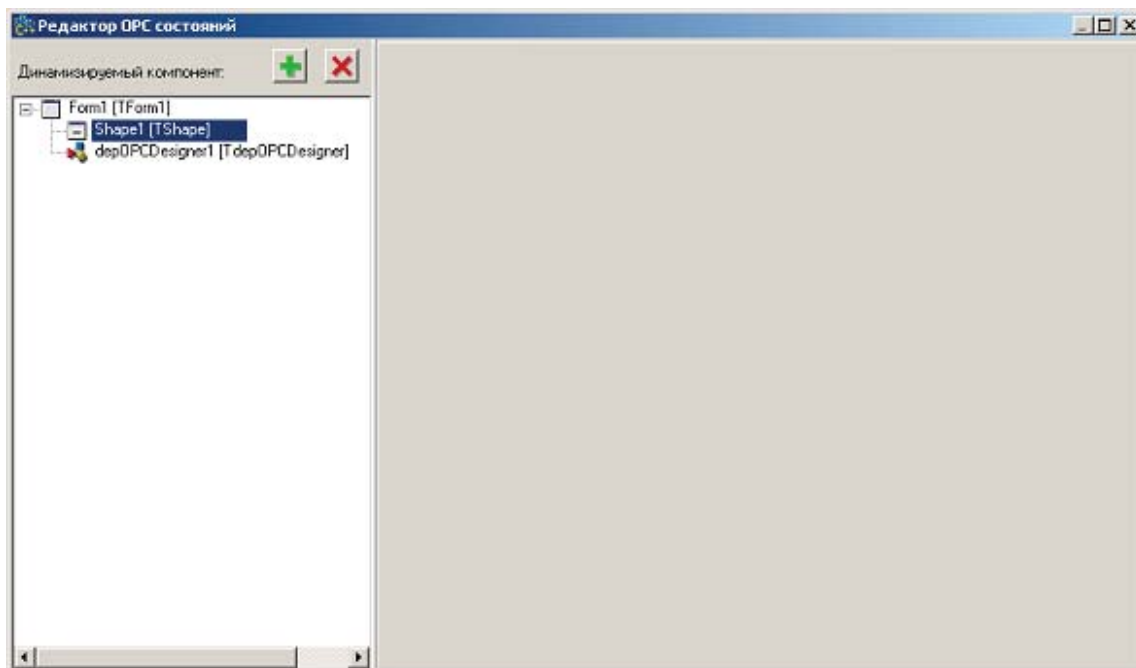



Рис. 6 Редактор OPC состояний.

Редактор разделен на две части. Справа находится дерево всех объектов, находящихся на форме, а справа окно привязок. Для того чтобы какому-либо объекту добавить привязку, надо выбрать (щелкнуть по нему мышкой) этот объект в дереве и

щелкнуть мышкой по значку "Добавить привязку" . При этом откроется окно со всеми доступными свойствами и событиями объекта, к которым можно привязаться.

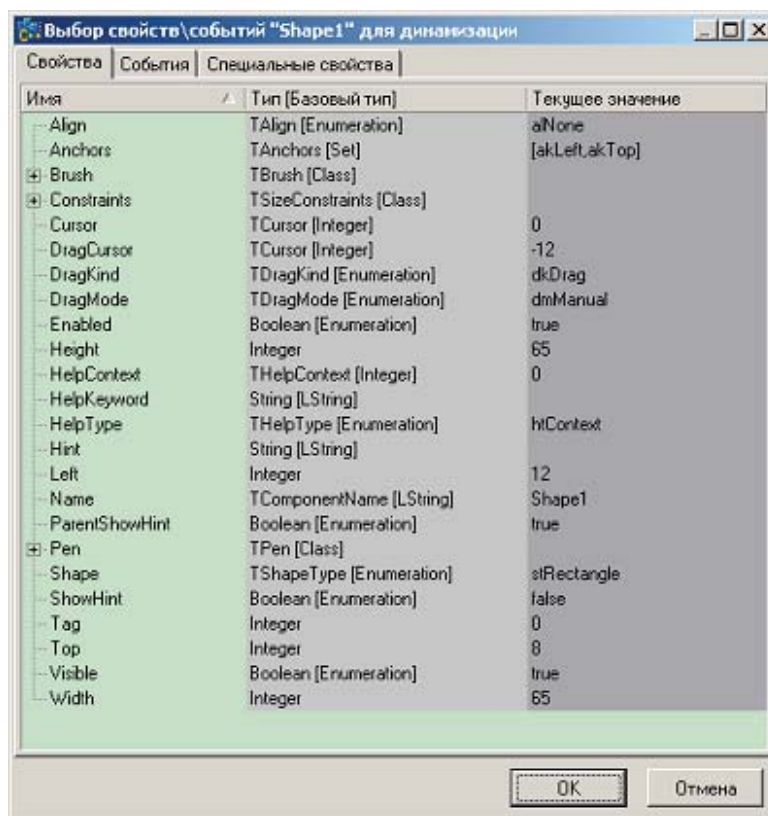


Рис. 7 Список свойств, доступных для привязки.

Выберите из списка свойство Brush | Color и нажмите "Ок".

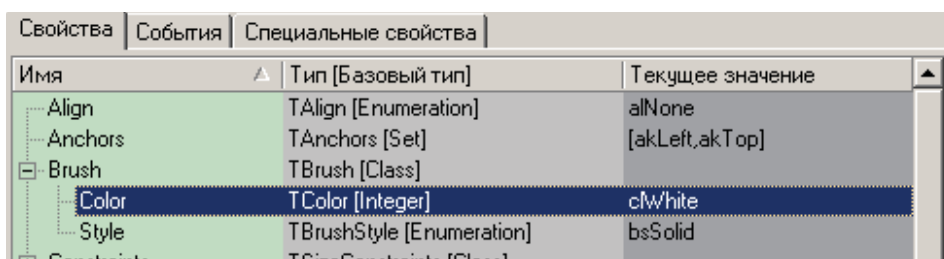


Рис. 8 Свойство цвета фона.

После этого, вид окна привязок для объекта Shape1 изменится.

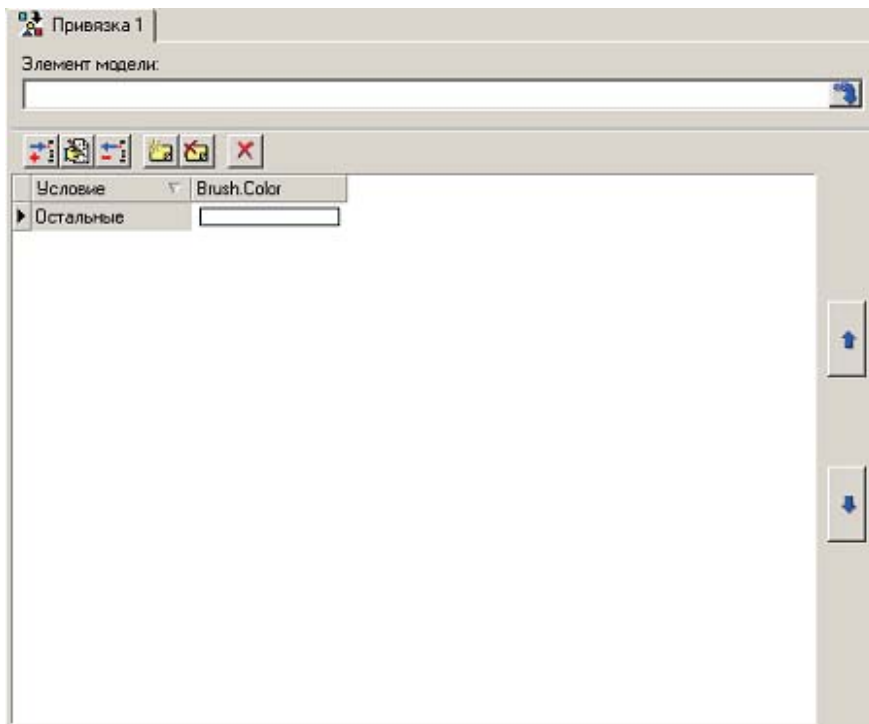



Рис. 9 Окно привязок для объекта Shape1.

Теперь надо указать элемент модели, к которому будем привязывать цвет фона. Щелкните мышкой по значку "Выбрать элемент модели" . Появится окно с деревом модели, в котором вам надо будет выбрать элемент для привязки. В нашем случае это состояние вентилятора (Fan | State).

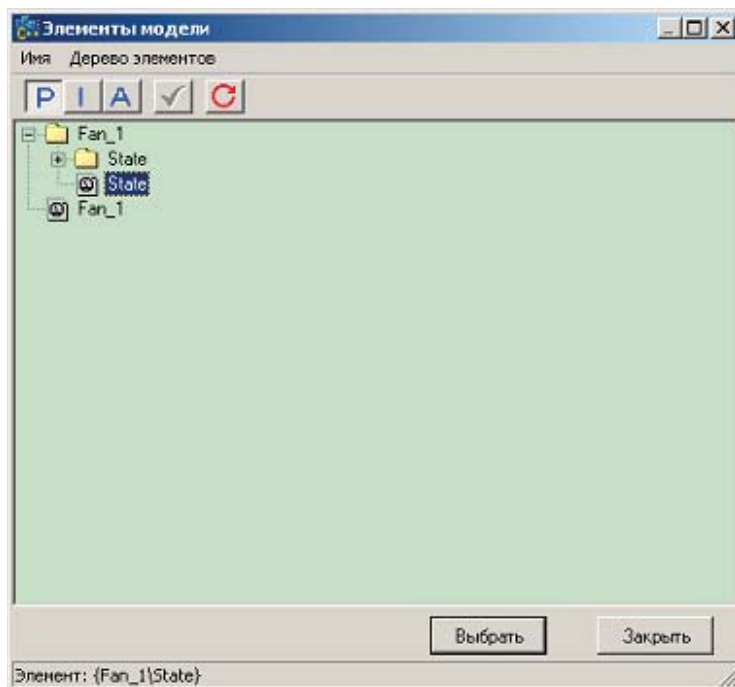


Рис. 10 Выбор элемента модели для привязки.

После этого в окне привязок для Shape1, в поле "Элемент модели", появится имя выбранного элемента.

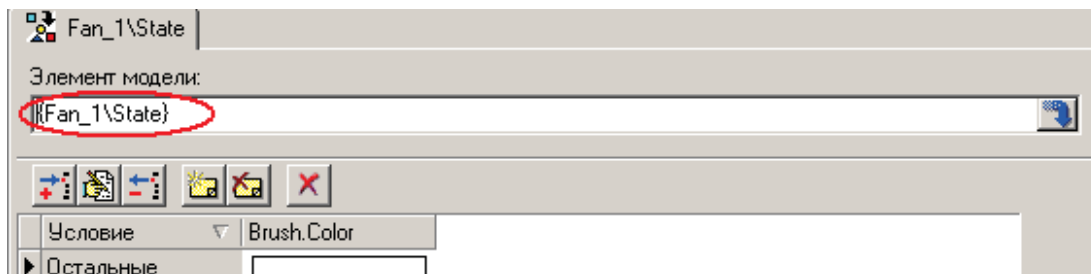



Рис. 11 Выбранный элемент модели.

Нам осталось описать зависимость цвета фона нашего объекта от значения элемента модели. Это делается с помощью панели инструментов для привязки свойств.



Рис. 12 Панель инструментов для привязки свойств.

Щелкните мышкой по значку "Добавить состояние" . В диалоге выберите свойство элемента модели, при котором должно происходить изменение объекта отображения. Нас интересует три состояния насоса: включен\выключен\не определено. Давайте добавим каждое из этих свойств.

При выключенном состоянии вентилятора, в дискрете состояния мы имеем 0.

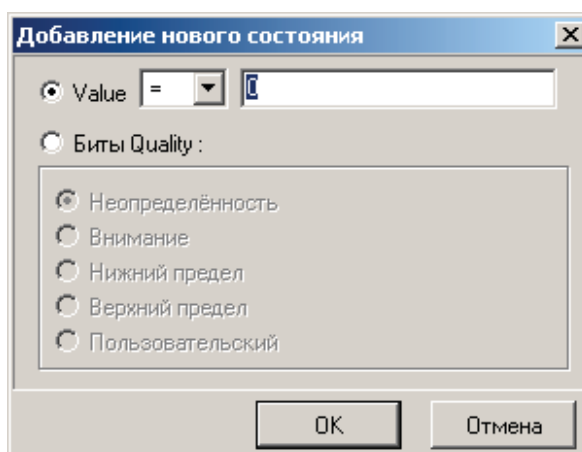


Рис. 13 Вентилятор отключен.

При включенном состоянии вентилятора, в дискрете состояния мы имеем 1.

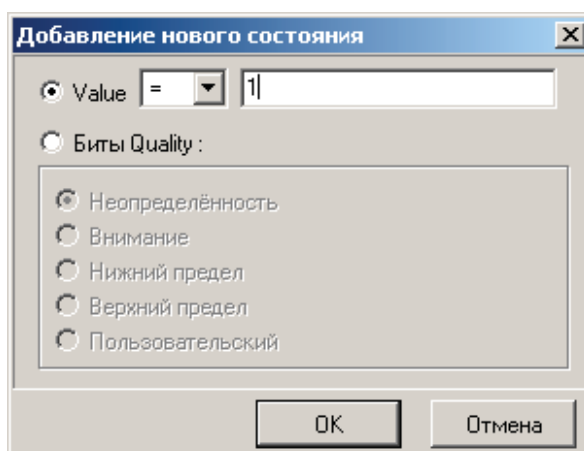


Рис. 14 Вентилятор включен.

Неопределенное состояние является следствием отсутствия связи с объектом и кодируется с помощью соответствующего бита Quality.

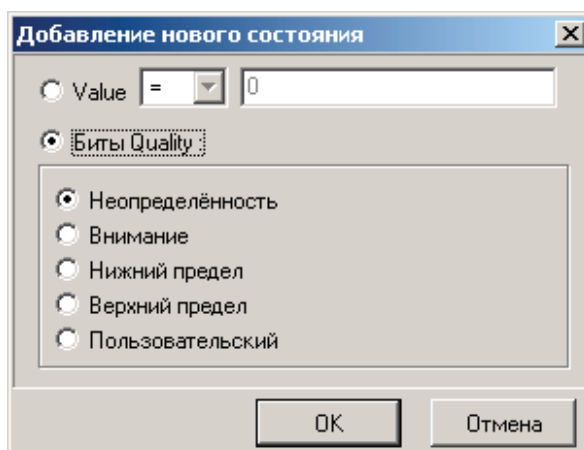


Рис. 15 Нет связи с объектом.

После добавления, соответствующие свойства появляются в окне привязок.

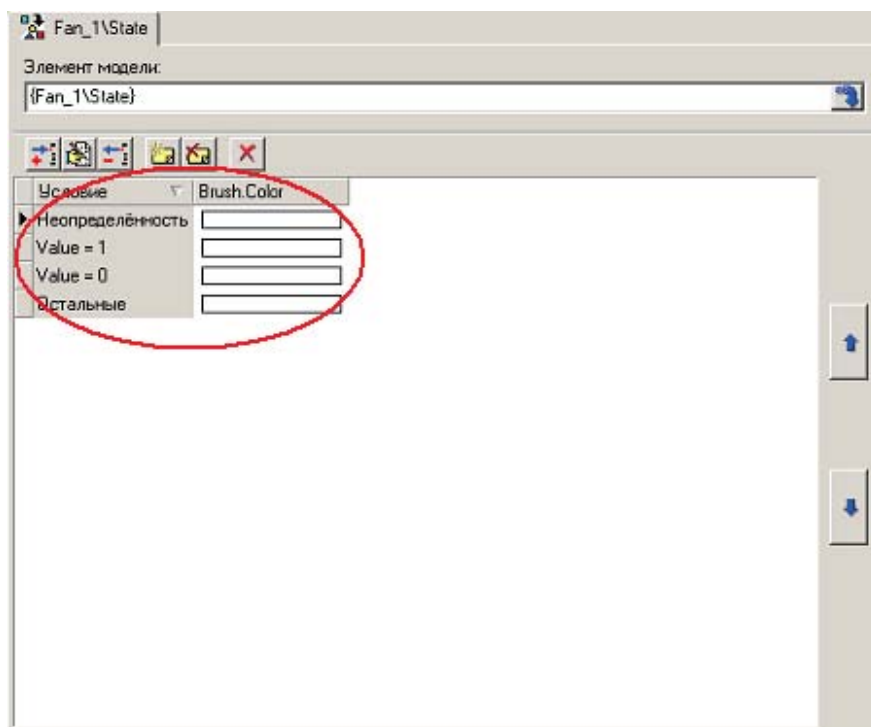


Рис. 16 Новые свойства в окне привязок.

Нам осталось выбрать цвет для каждого из состояний элемента модели (вентилятора). Пусть включенное состояние вентилятора отображается зеленым цветом, выключенное - серым, неопределенное - желтым. Для выбора цвета щелкните мышкой по соответствующей ячейке в столбце "Brush.Color" и выберите цвет из списка.

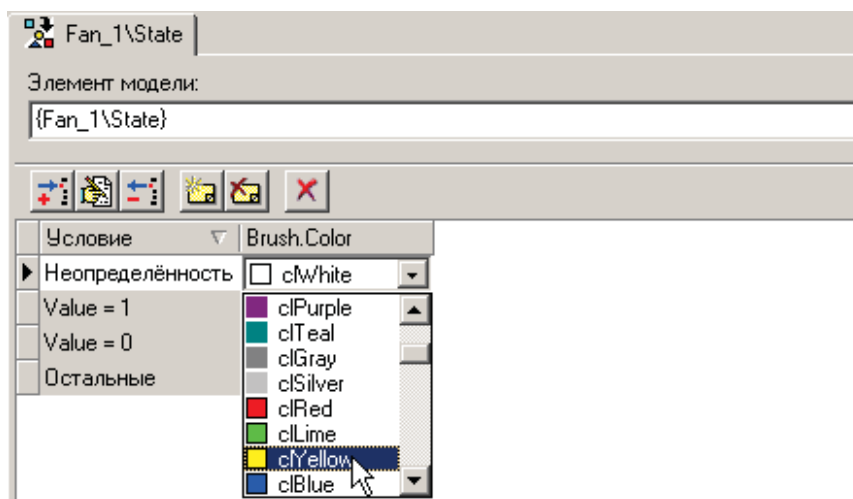


Рис. 17 Выбор цвета.

После всех операций окно привязок должно иметь следующий вид.

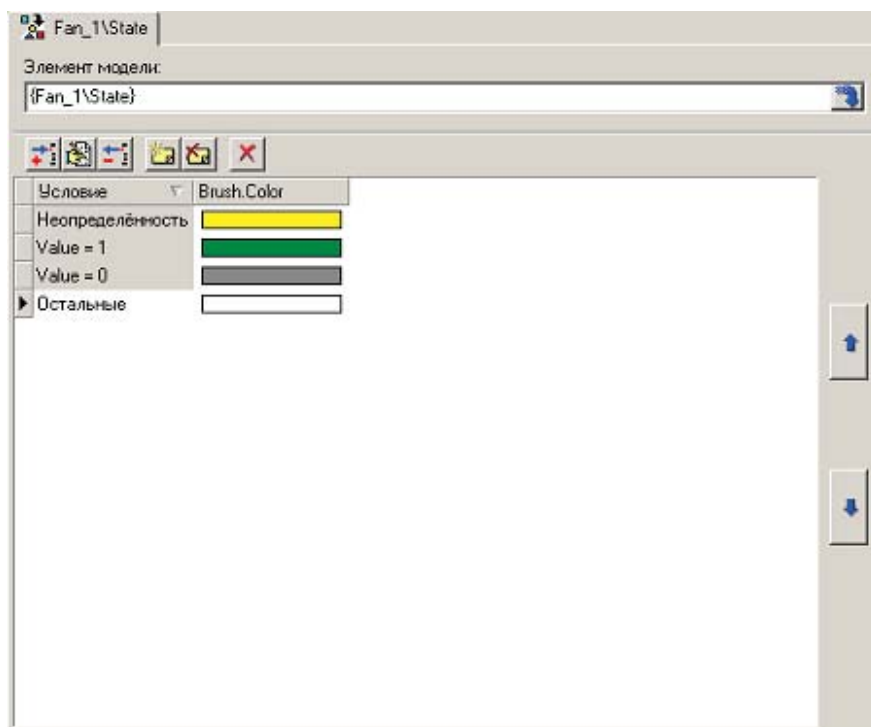


Рис. 18 Окно привязок после добавления всех свойств.

На этом добавление свойств заканчивается.

Теперь вы обладаете необходимыми знаниями для того, чтобы, сделать отображение для нашей первой модели.

Следующий шаг: [Ваше первое отображение.](#)

9.4.2.4 Ваше первое отображение

Все готово к созданию нашего первого отображения. Сначала мы должны нарисовать наше отображение и определиться, какие компоненты мы будем использовать.

Пусть наше отображение должно иметь следующий вид (рис. 1).



Рис. 1 Внешний вид отображения.

Теперь давайте опишем поведение нашего отображения.

1. Выключенное состояние вентилятора характеризуется светло-серым цветом фона окружности.
2. Включенное состояние вентилятора характеризуется зеленым цветом фона окружности.
3. Неопределенное состояние характеризуется желтым цветом окружности.

Создадим новый проект. В среде ++Builder войдите в меню File | New | Other и в закладке DEP выберите depOPCProject.

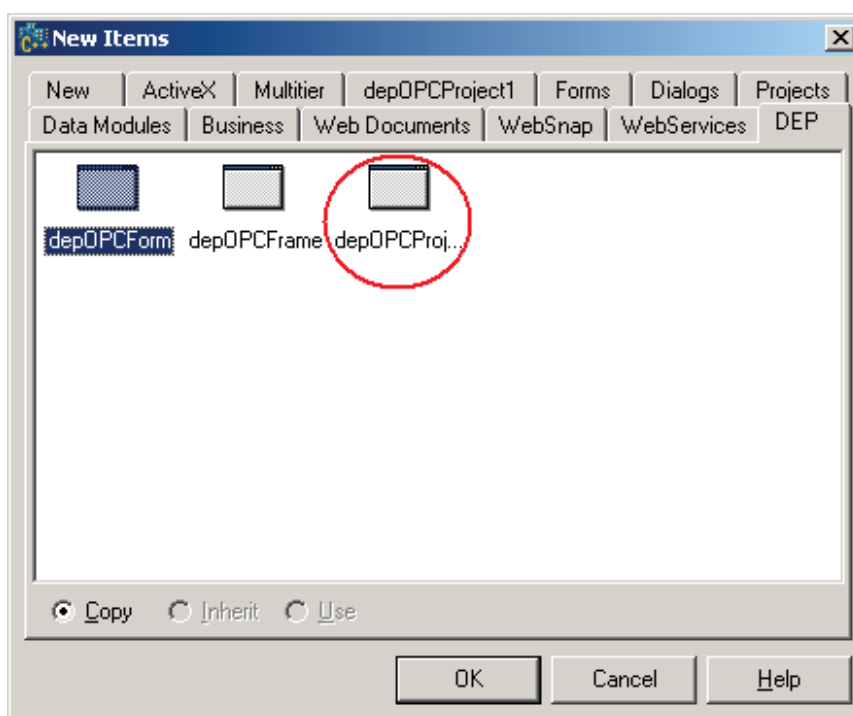


Рис. 2 Создание нового проекта.

После этого на экране появится новая форма, на которой уже будет находиться объект depOPCDesigner.

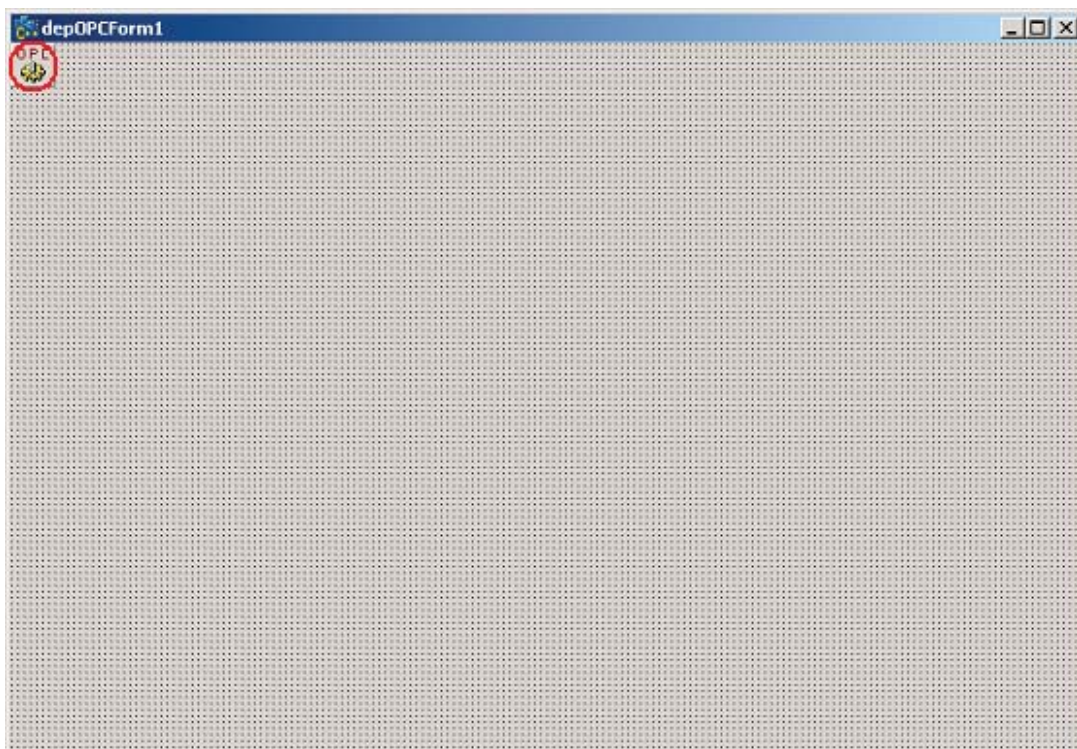


Рис. 3 Новая форма с объектом depOPCDesigner.

Уменьшите размеры формы и положите на нее два объекта DEPShare (закладка DEP).

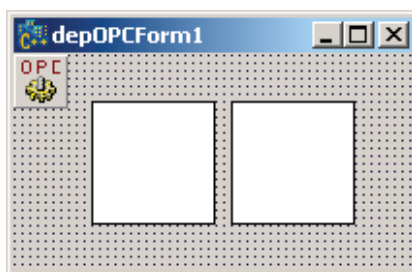


Рис. 4 Форма с двумя объектами DEPShare.

Измените форму (свойство Shape Type) одного объекта на круг, а другого на треугольник.

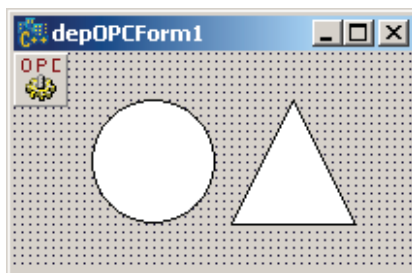


Рис. 5 Изменение формы объектов.

Измените ориентацию треугольника (свойство Orientation) и совместите два объекта.

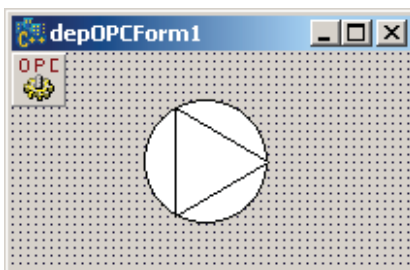


Рис. 6 Совмещение двух объектов.

Пришла пора привязать свойство цвета фона круга к элементу модели Fan | State. Выделите объект в виде круга и щелкните правой кнопкой мыши. В выпадающем меню выберите "Редактор OPC состояний".

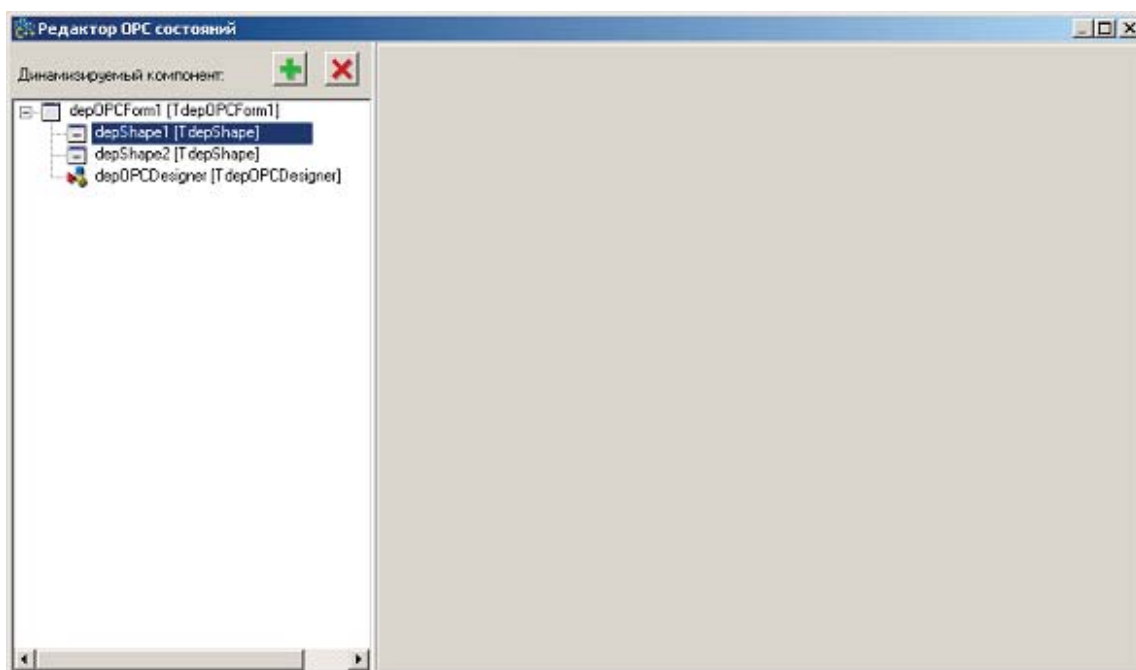


Рис. 7 Редактор OPC состояний.

Щелкните мышкой "Добавить привязку" . Выберите свойство Brush | Color и нажмите "ОК".

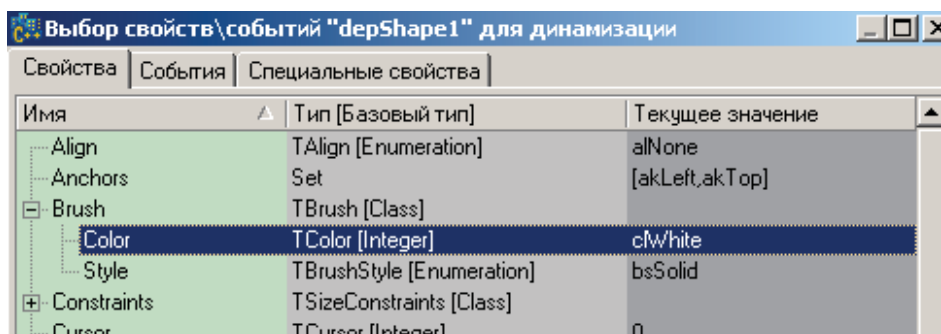


Рис. 8 Выбор свойства.

В окне редактирования привязок выберите элемент модели (щелкните мышкой по  Fan | State).

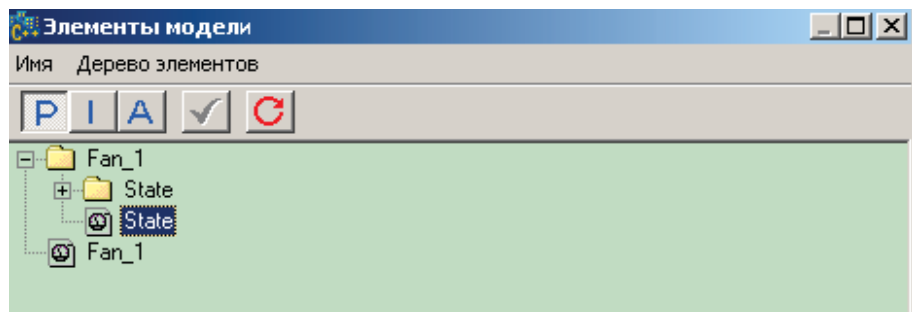


Рис. 9 Выбор элемента модели.

Добавьте состояния элемента модели (щелкните мышкой по ): включенное, выключенное и неопределенное.

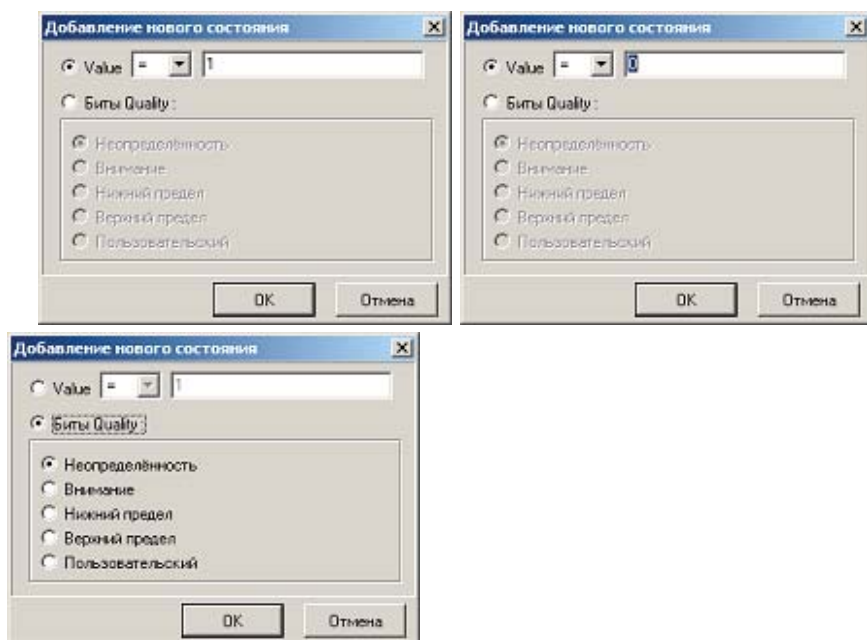


Рис. 10 Добавление включенного, выключенного и неопределенного состояний.

Выберите цвет для каждого из состояний в соответствии с заданием (см. выше).

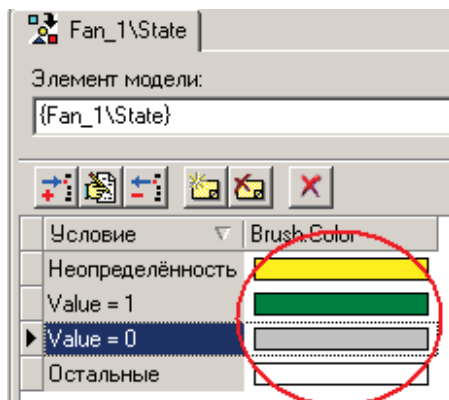


Рис. 11 Выбор цвета для каждого состояния элемента модели.

Теперь нам осталось сохранить и собрать проект, после чего можно будет его запустить и посмотреть, что у нас получилось.

Сохраните (меню File | Save All) и соберите проект (Ctrl+ F9). Если во время сборки компилятор не нашел ни одной ошибки, значит вы все сделали правильно и настал торжественный момент запуска вашего первого отображения. Нажмите F9 (меню Run | Run).

Если модель у вас запущена вы увидите следующую картину.

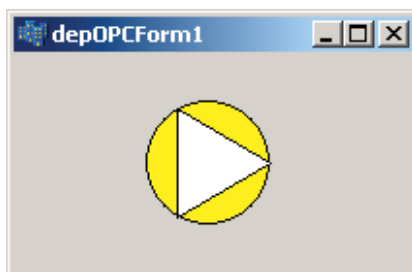


Рис. 12 Неопределенное состояние вентилятора.

Круг имеет желтый цвет фона, потому что значение состояния вентилятора неопределенно. Для того чтобы проверить работу нашего отображения мы должны проимитировать работу датчика состояния вентилятора. Для этого мы должны записать в первый дискрет базы параметров WinDecont 1 (включенное состояние) или 0 (выключенное состояние). При этом вы увидите, как круг меняет цвет своего фона.

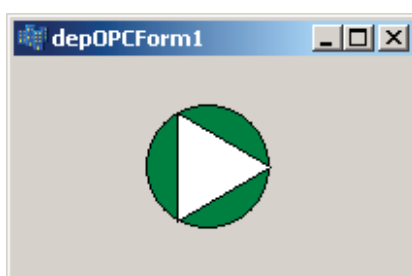


Рис. 13 Вентилятор работает.

Поздравляю! Вы только что сделали свое первое приложение, отражающее состояние вашего вентилятора!

Следующий шаг: [Итоги](#).

9.4.2.5 Итоги

Итогами нашей работы стало законченное отображение состояния вентилятора. Мы сделали модель и отображение, и связали все это вместе с помощью "ОРС-дизайнера". Теперь, выполнив эту первую, небольшую, но важную работу, вы готовы двигаться дальше и решать уже более сложные задачи. Давайте укрепим наши знания, решив задачу по проектированию и построению АРМ диспетчера для небольшого технологического объекта.

Следующий шаг: [Построение АРМ диспетчера](#).

9.4.3 Построение АРМ диспетчера. Пример 1.

9.4.3.1 Введение

Давайте, в начале, определимся, из каких этапов будет состоять наша работа.

1. Описание структуры объекта, его входных и выходных сигналов.
2. Написать список требований к АРМ (какая информация должна быть отображена, алгоритмы работы, аварии и квитация и т.п.).
3. Построить модель.
4. Построить отображение.
5. Связать отображение с моделью.
6. Тестирование.

Приведенное разбиение не является единственно верным, а лишь структурирует основные этапы работы, с целью разделения одной большой задачи на несколько меньших подзадач.

Начнем нашу работу с описания структуры объекта и списка сигналов.

Следующий шаг: [Структура объекта](#).

9.4.3.2 Структура объекта

В качестве примера, рассмотрим упрощенную структуру центрального теплового пункта (ЦТП).

Структура объекта

Пусть ЦТП содержит две группы насосов:

1. Группа насосов горячей воды (2 насоса).
2. Группа насосов отопления (2 насоса).

Далее предположим, что для каждой группы имеются следующие характеристики:

1. Входное и выходное давления.
2. Режим работы (местный или дистанционный), определяющий разрешение на удаленное управление.
3. Каждый из насосов в группе обладает:
 - 3.1 Состоянием включен\выключен.
 - 3.2 Импульсными командами на включение и выключение.

Пусть также, ЦТП производит мониторинг параметров:

1. Наличие пожарной сигнализации.
2. Контроль напряжения.

Исходя из предложенной структуры, мы имеем следующий набор сигналов.

Набор сигналов

- Режим работы насосов горячей воды.
- Состояние насоса горячей воды 1.
- Состояние насоса горячей воды 2.
- Режим работы насосов отопления.
- Состояние насоса отопления 1.
- Состояние насоса отопления 2.
- Пожарная сигнализация.
- Напряжение.

Телеуправление (ТУ)

- Включить насос горячей воды 1.
- Отключить насос горячей воды 1.
- Включить насос горячей воды 2.
- Отключить насос горячей воды 2.
- Включить насос отопления 1.
- Отключить насос отопления 1.
- Включить насос отопления 2.
- Отключить насос отопления 2.

Телеизмерение (ТИ)

- Давление горячей воды на входе.
- Давление горячей воды на выходе.

- Давление отопления на входе.
- Давление отопления на выходе.

Следующий шаг: [Список требований](#).

9.4.3.3 Список требований

Пусть мы имеем следующий список требований к АРМ.

1. Работать непрерывно в режиме реального времени под управлением операционной системы Windows 98 или Windows 2000.
2. Отображать технологическую схему и текущие технологические параметры ЦТП.
3. Отображать индикацией работающее и неработающее оборудование, а также оборудование в неопределенном состоянии.
4. Производить управление оборудованием ЦТП.

Описание технологического экрана (отображения)

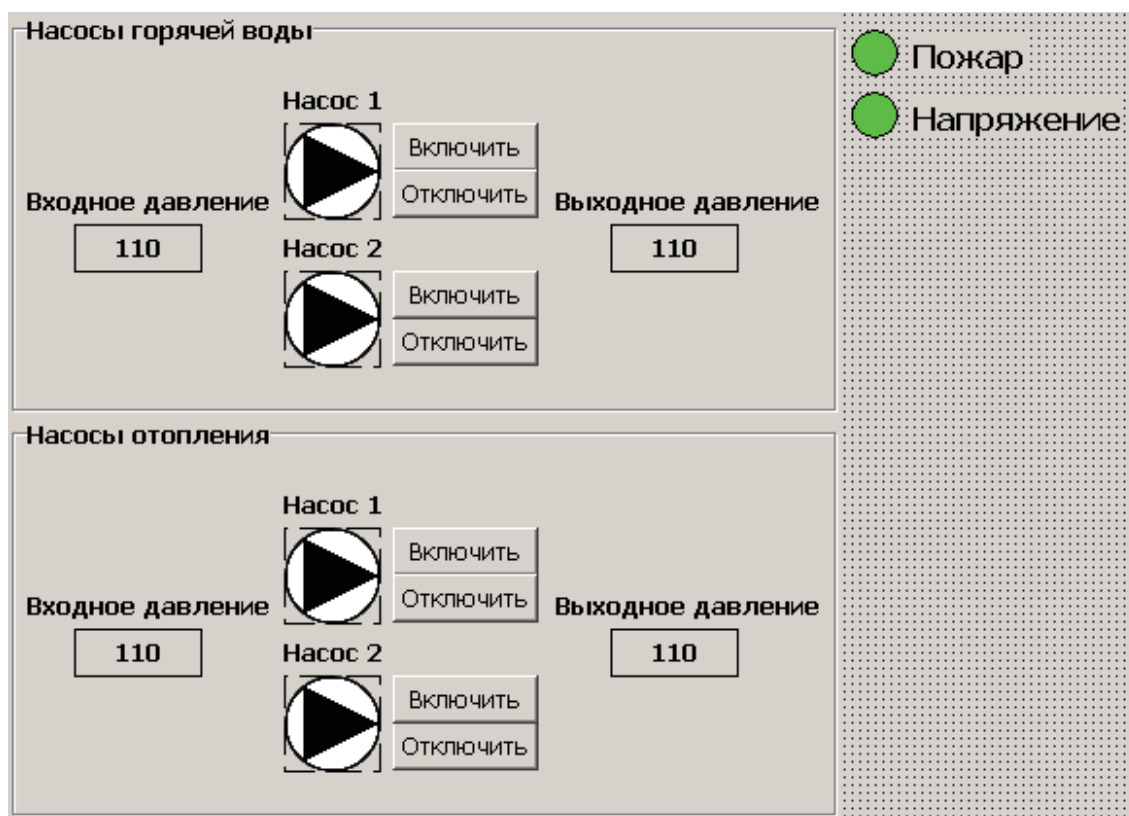


Рис. 1 Вид технологического экрана.

Насосы

Работающий насос имеет зеленый цвет фона круга, неработающий белый. Если состояние насоса неопределенно фон круга имеет желтый цвет.

Кнопка "Включить" посылает импульсную команду на включение насоса.

Кнопка "Отключить" посылает импульсную команду на отключение насоса.

Давления

В метках выводятся значения входного и выходного давлений.

Пожар

При срабатывании датчика пожарной сигнализации круг становится красного цвета, в нормальном состоянии зеленого. Если значение сигнала от датчика неопределенно круг имеет желтый цвет.

Напряжение

При пропадании напряжения на ЦТП круг становится красного цвета, в нормальном состоянии зеленого. Если значение сигнала от датчика неопределенно круг имеет желтый цвет.

Следующий шаг: [Создание модели](#).

9.4.3.4 Создание модели

9.4.3.4.1 Создание нового проекта

Начнем построение модели с создания нового проекта.

Запустите "Конструктор OPC-модели".

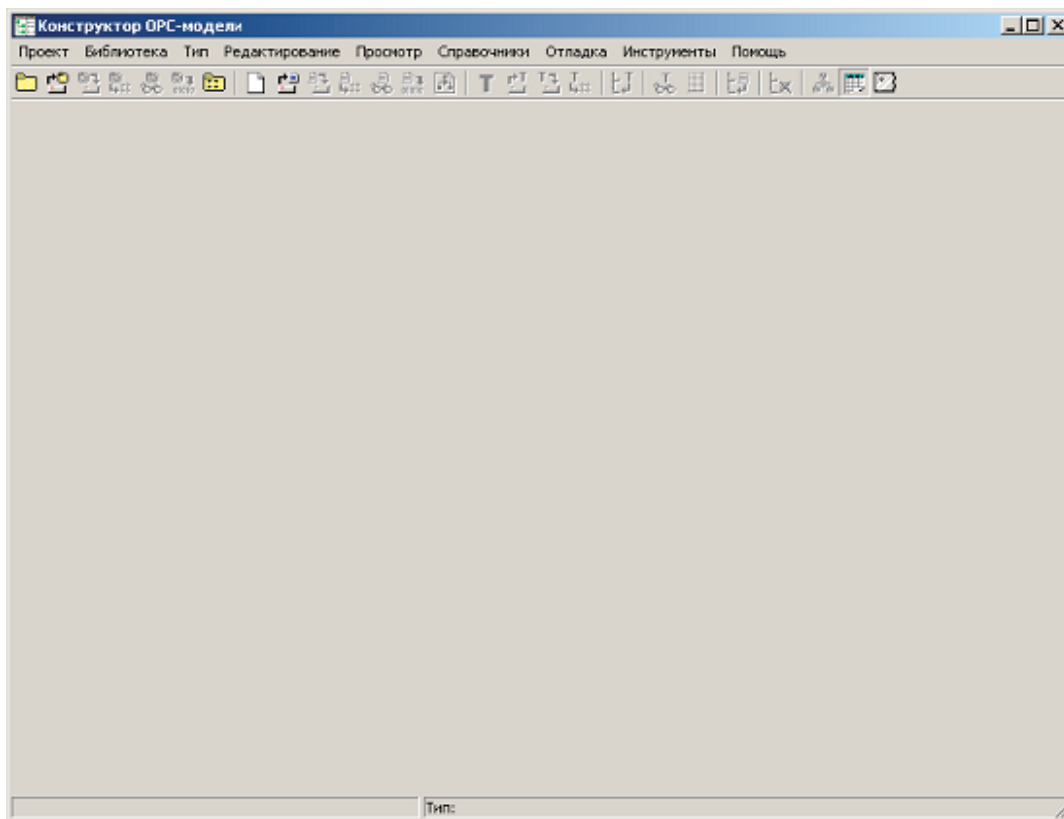


Рис. 1 Конструктор OPC-модели.

Создайте новый проект () и сохраните его (). Библиотеку назовите ARMLib, а проект ARM.

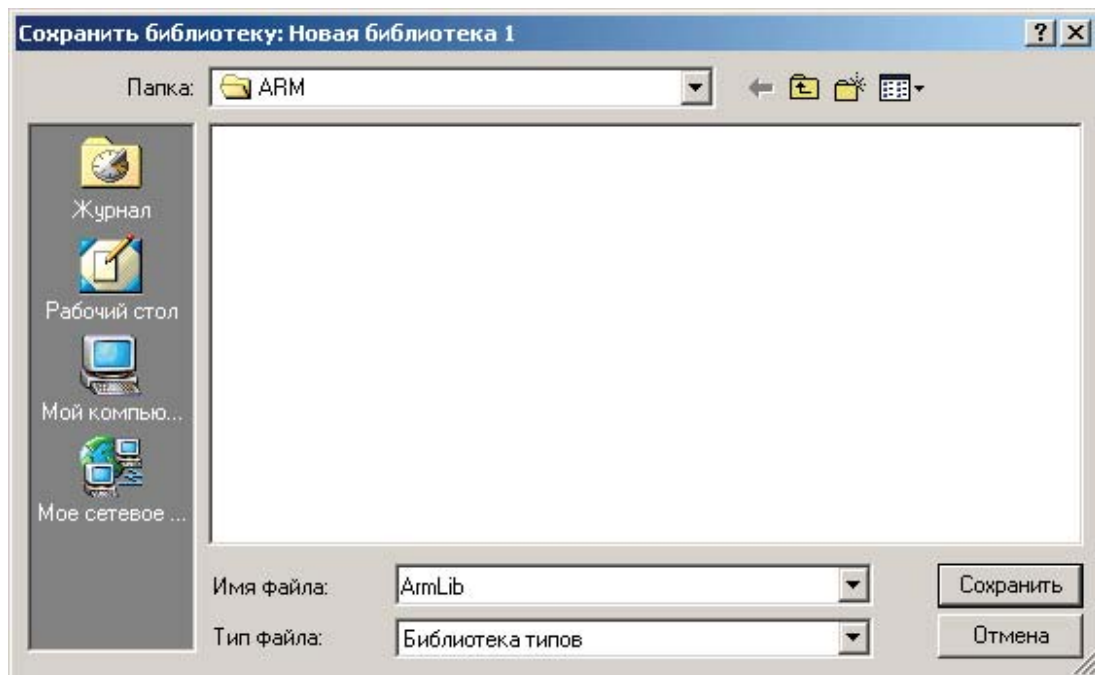


Рис. 2 Сохранение библиотеки.

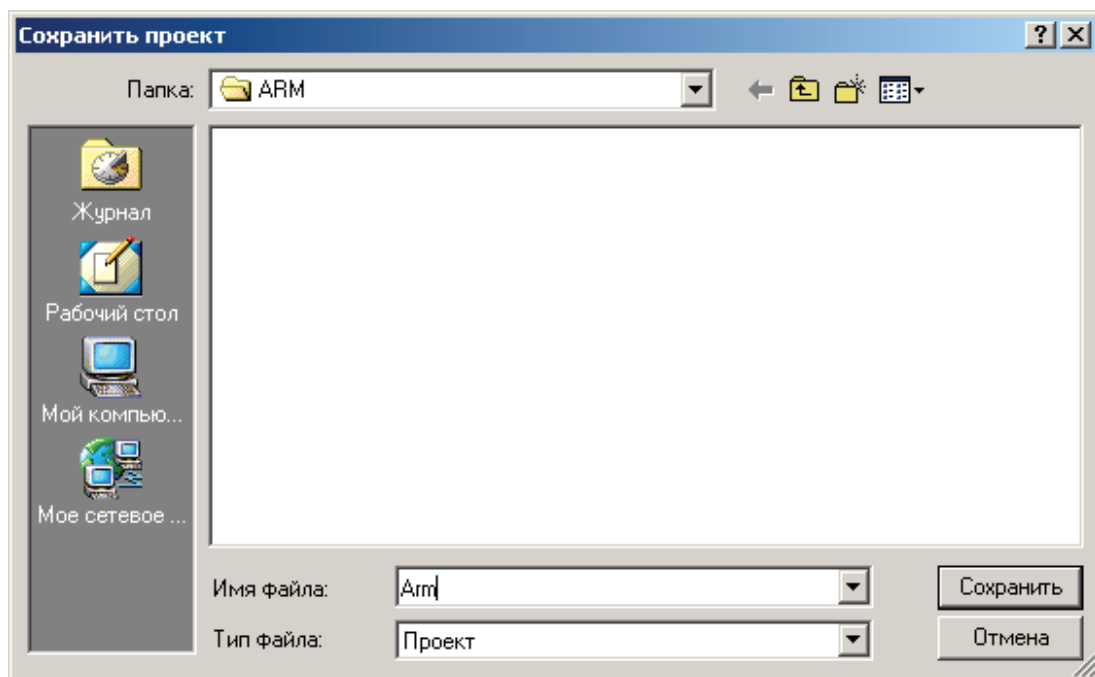


Рис. 3 Сохранение проекта.

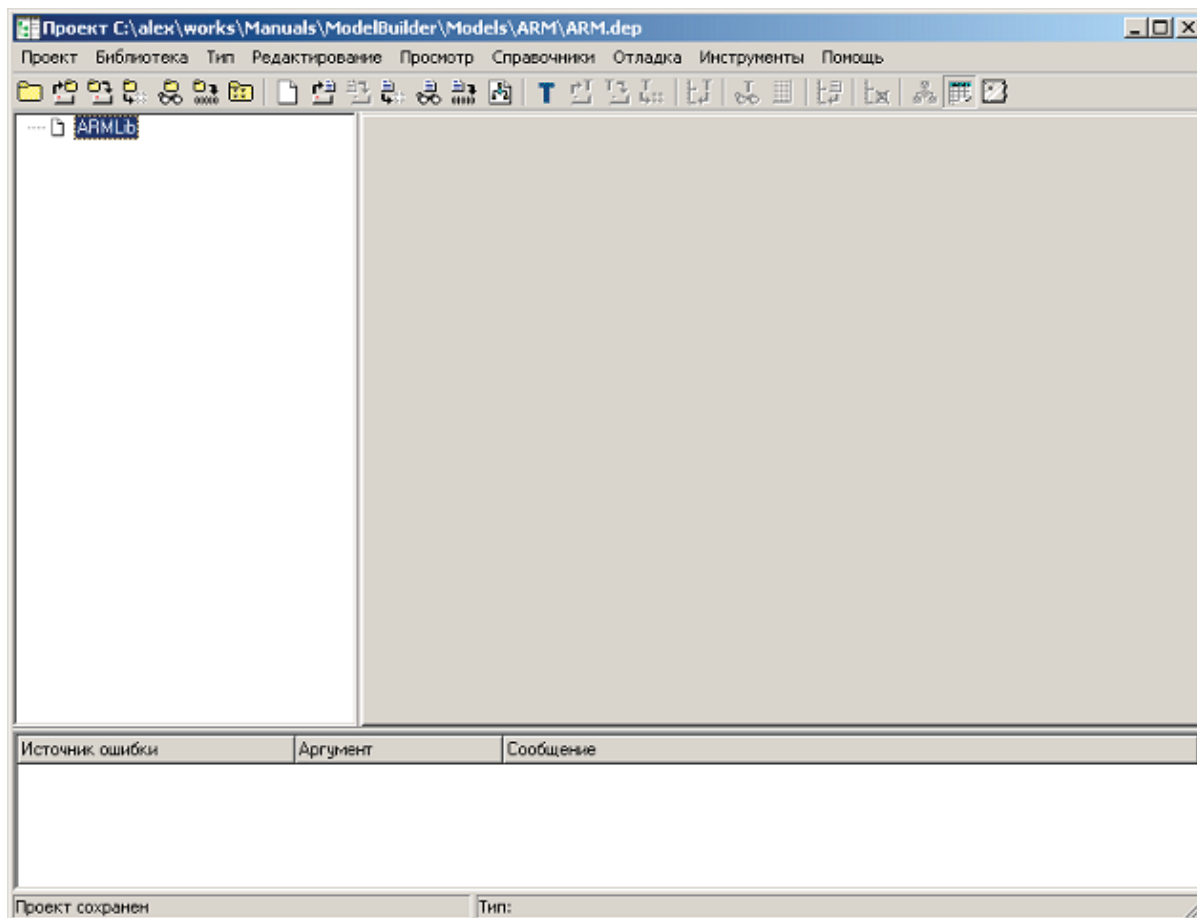


Рис. 4 Новый проект.

Давайте теперь подумаем, какие нам потребуются типы и как нам надо будет их структурировать.

Следующий шаг: [Структура модели.](#)

9.4.3.4.2 Структура модели

Исходя из структуры объекта ([Структура объекта](#)) нам потребуются следующие типы:

Тип - насос (Pump)

Насос имеет состояние и управление (включить\выключить). Состояние насоса это дискретный входной сигнал, а управление это дискретный импульсный выходной сигнал. Исходя из этого, построим структуру типа насос (как мы помним, в названиях типов и элементов используются только латинские символы).

Pump (насос)

- State (состояние) - входной дискрет.
- On (включить) - выходной импульсный дискрет.
- Off (выключить) - выходной импульсный дискрет.

Насос не имеет сценариев работы.

Тип - группа насосов (PumpGroup)

Группа насосов имеет два насоса, входное и выходное давления, а также режим управления. Входное и выходное давления это входные аналоговые сигналы. Режим управления это входной дискретный сигнал.

PumpGroup (группа насосов)

- Pump_1 (насос 1) - тип Pump.
- Pump_2 (насос 2) - тип Pump.
- InputPressure (входное давление) - входной аналог.
- OutputPressure (выходное давление) - входной аналог.
- Mode (режим управления) - входной дискрет.

Группа насосов не имеет сценариев работы.

Главный тип - ЦТП (СТР)

ЦТП имеет две группы насосов, а также ведет контроль пожарной сигнализации и напряжения. Пожарная сигнализация и напряжение это входные дискретные сигналы.

СТР (ЦТП)

- HotWaterGroup (группа насосов горячей воды) - тип PumpGroup.
- HeatingGroup (группа насосов отопления) - тип PumpGroup.
- Fire (пожарная сигнализация) - входной дискрет.
- Voltage (напряжение) - входной дискрет.

ЦТП не имеет сценариев работы.

Нам надо теперь создать все эти типы в "Конструкторе OPC-модели".

Следующий шаг: [Создание типа - насос.](#)

9.4.3.4.3 Создание типа - насос

Создайте новый тип (). Назовите его Pump, выберите Int в качестве базового типа.

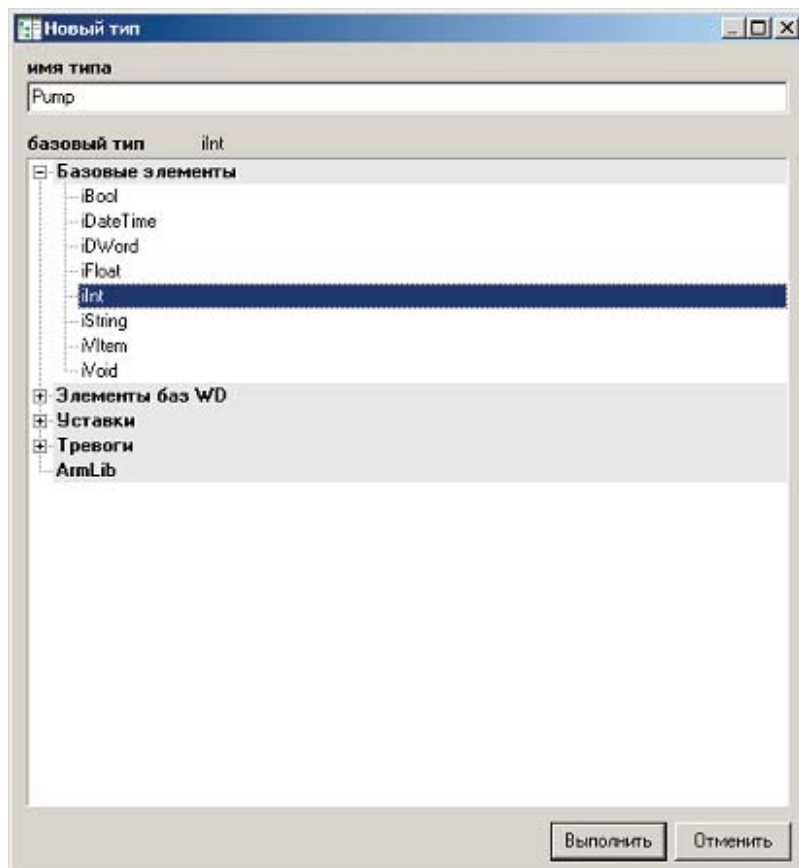


Рис. 1 Создание нового типа.


Добавьте типу новый элемент (). Назовите его State (Состояние). Как вы помните, состояние это входной дискрет, поэтому, в качестве базового типа выбираем wdDip из библиотеки "Элементы баз Wd".



Рис. 2 Добавление свойства State (состояние насоса).

Добавьте свойства управления (включение и выключение). Так как включение и выключение это выходные импульсные дискреты в качестве базового типа выбираем wdOutImp из библиотеки "Элементы баз WD".

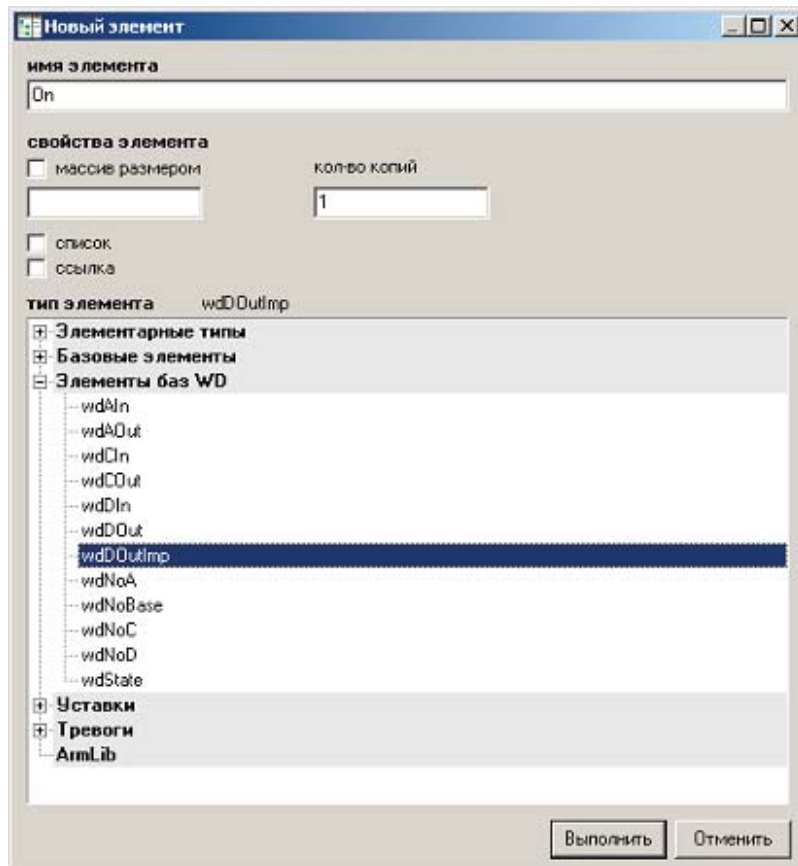


Рис. 3 Добавление свойства On (включение насоса).

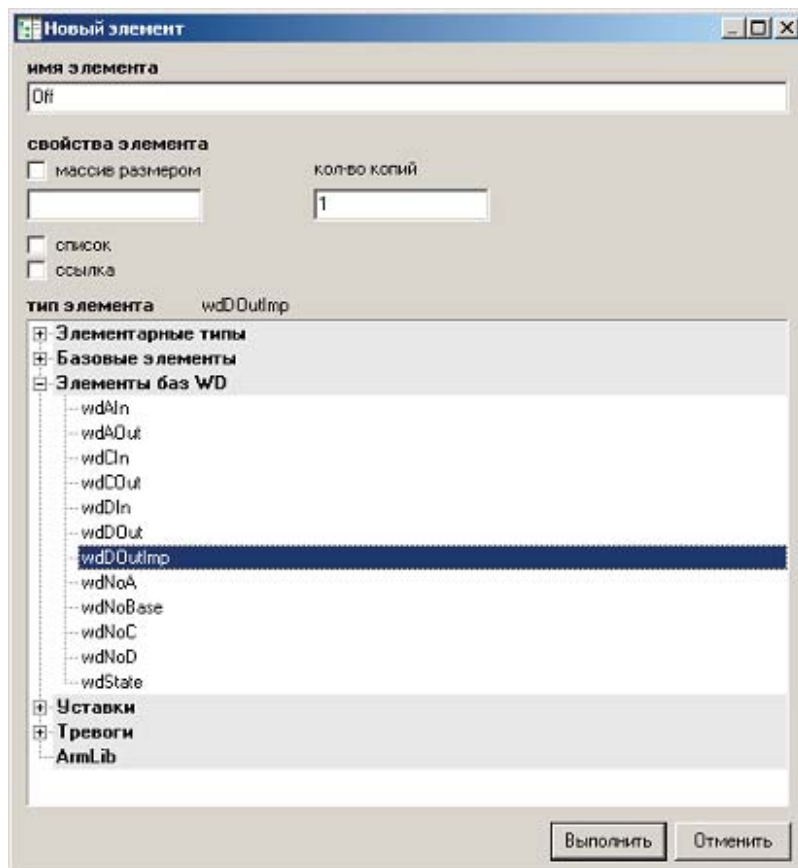


Рис. 4 Добавление свойства Off (выключение насоса).

После всех операций окно редактирования типа примет вид:

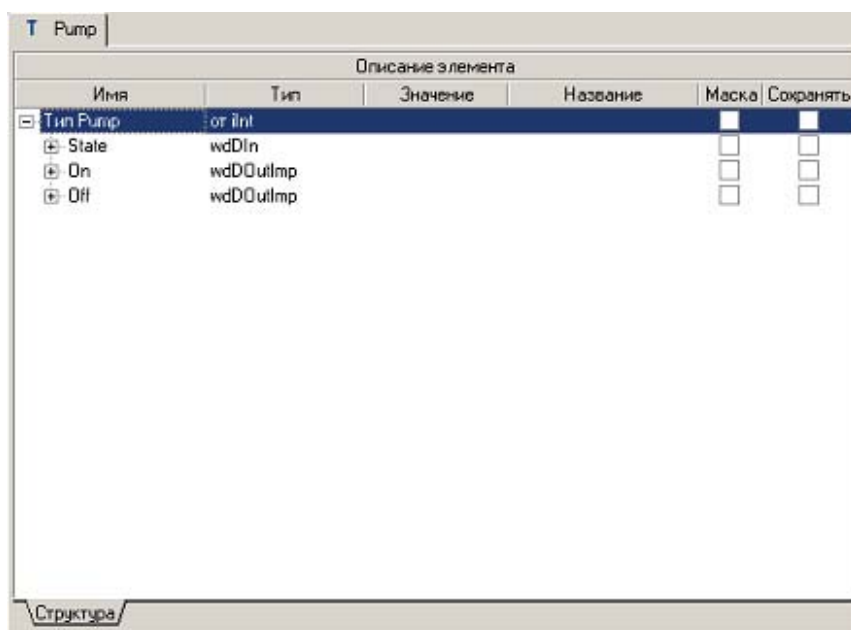


Рис. 5 Тип Pump.

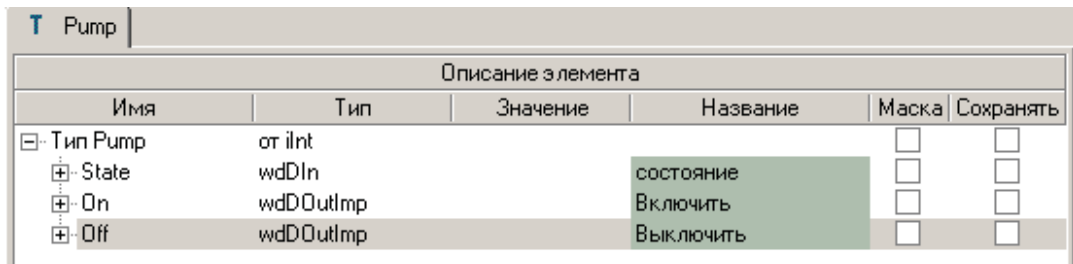
Дадим название всем свойствам. Для этого щелкните мышкой по ячейке в строке Pump.state в столбце "Название". Вокруг ячейки при этом появится пунктирная рамочка. Напишите "состояние".



Описание элемента					
Имя	Тип	Значение	Название	Маска	Сохранять
[-] Тип Pump	ot ilnt			<input type="checkbox"/>	<input type="checkbox"/>
[-] State	wdDIn		состояние	<input type="checkbox"/>	<input type="checkbox"/>
[-] On	wdDOutImp			<input type="checkbox"/>	<input type="checkbox"/>
[-] Off	wdDOutImp			<input type="checkbox"/>	<input type="checkbox"/>

Рис. 6 Редактирование названия для свойства State.

Аналогичным образом впишите названия для On - "Включить" и Off - "Выключить".



Описание элемента					
Имя	Тип	Значение	Название	Маска	Сохранять
[-] Тип Pump	ot ilnt			<input type="checkbox"/>	<input type="checkbox"/>
[-] State	wdDIn		состояние	<input type="checkbox"/>	<input type="checkbox"/>
[-] On	wdDOutImp		Включить	<input type="checkbox"/>	<input type="checkbox"/>
[-] Off	wdDOutImp		Выключить	<input type="checkbox"/>	<input type="checkbox"/>

Рис. 7 Тип Pump с названиями элементов.

Сохраните тип Pump (). Тип Pump появится в нашей библиотеке.



Рис. 8 Добавление типа Pump в библиотеку.

Следующий шаг: [Создание типа - группа насосов.](#)

9.4.3.4.4 Создание типа - группа насосов

Создайте новый тип, назовите его PumpGroup. В качестве базового типа выберите ilnt.

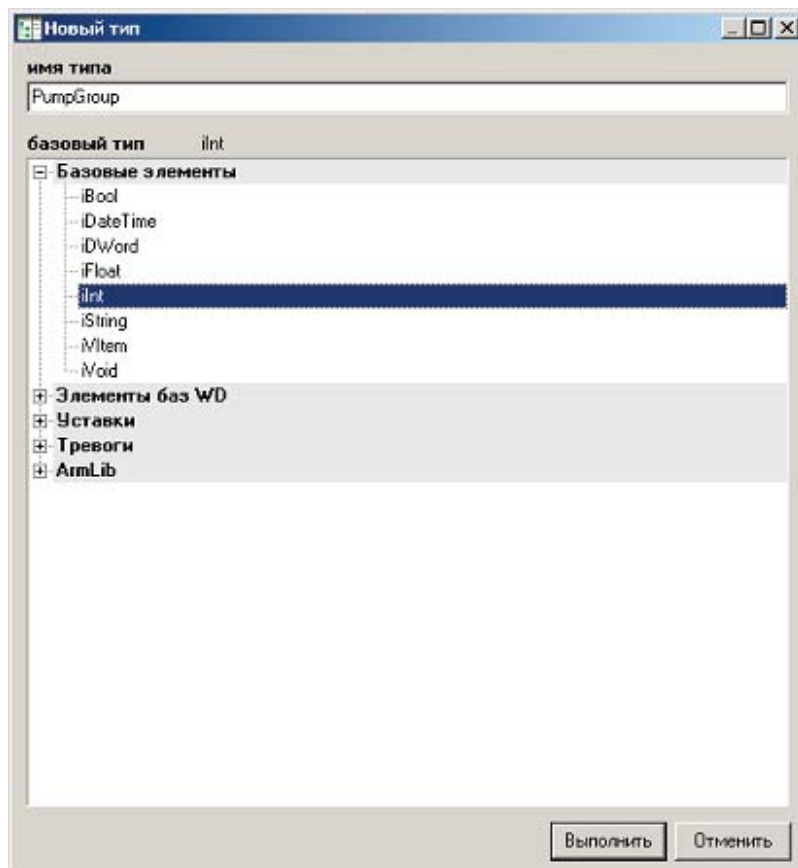


Рис. 1 Создание типа PumpGroup (группа насосов).

Добавьте типу элементы: Pump_1 (насос 1) - тип Pump (библиотека "ArmLib"), Pump_2 (насос 2) - тип Pump (библиотека "ArmLib"), InputPressure (входное давление) - тип wdAIIn (библиотека "Элементы баз WD"), OutputPressure (выходное давление) - тип wdAIIn (библиотека "Элементы баз WD"), Mode (режим работы) - тип wdDIIn (библиотека "Элементы баз WD").

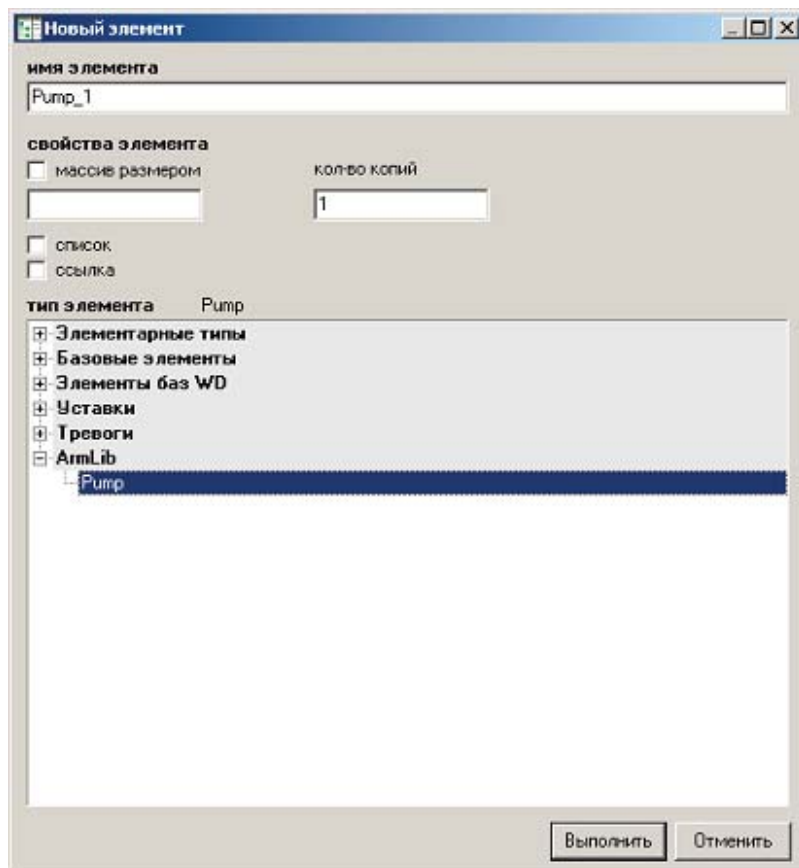


Рис. 2 Добавление свойства Pump_1 (насос 1).

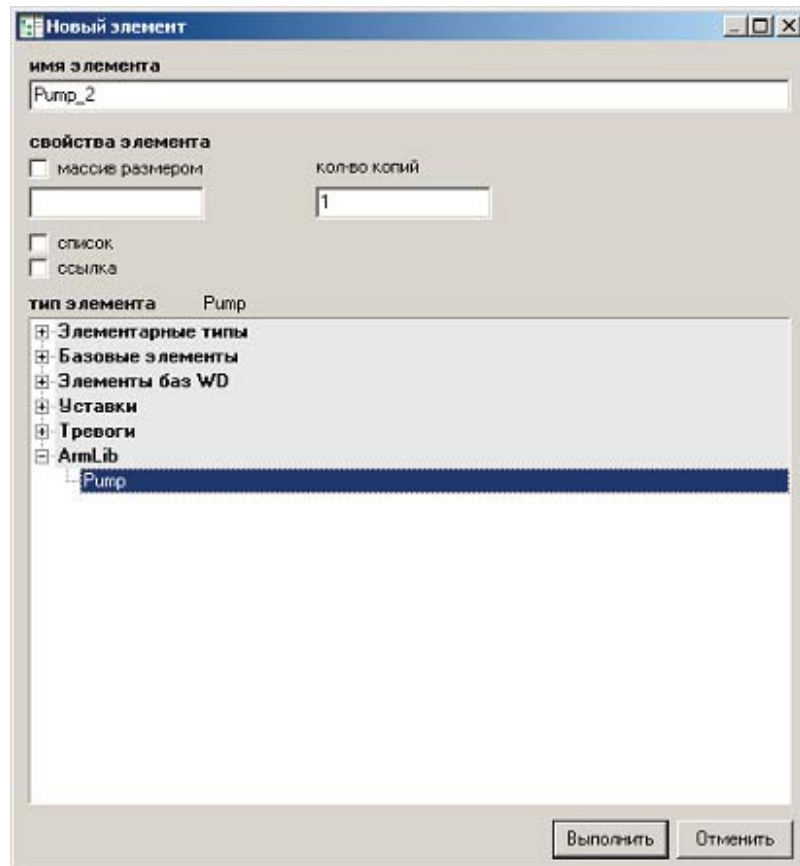


Рис. 3 Добавление свойства Pump_2 (насос 2).



Рис. 4 Добавление свойства InputPressure (входное давление).

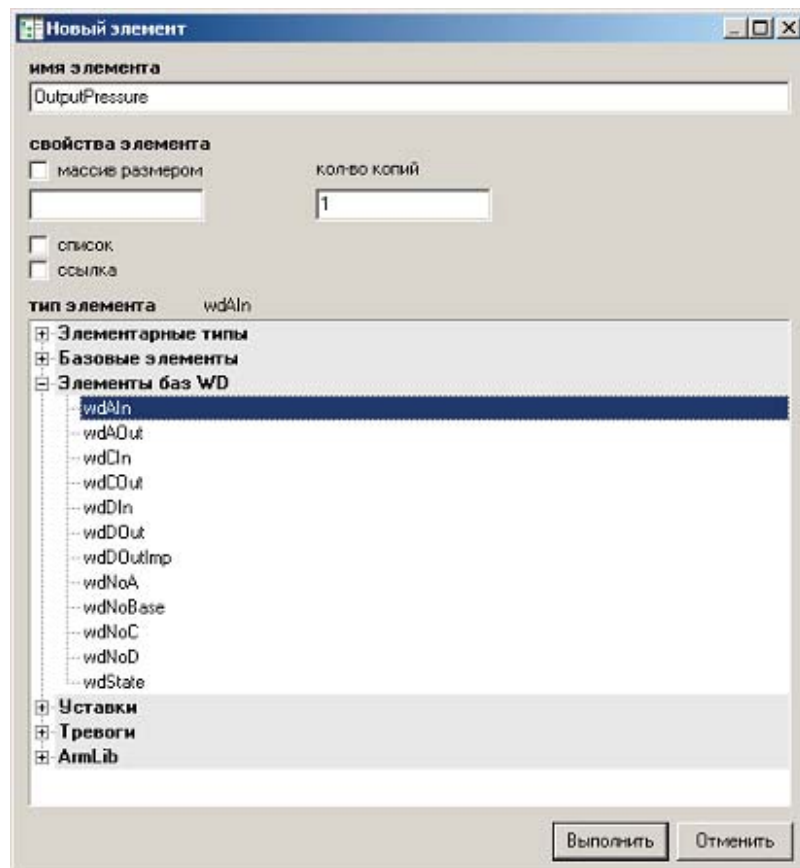


Рис. 5 Добавление свойства OutputPressure (выходное давление).

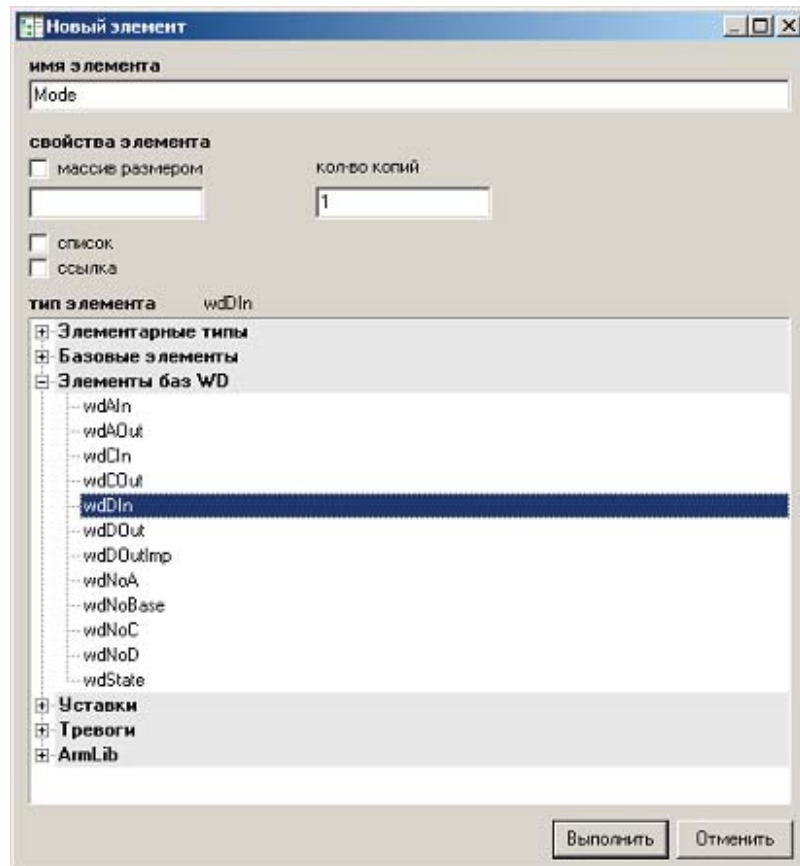


Рис. 6 Добавление свойства Mode (режим работы).

Окно редактирования типа примет вид:

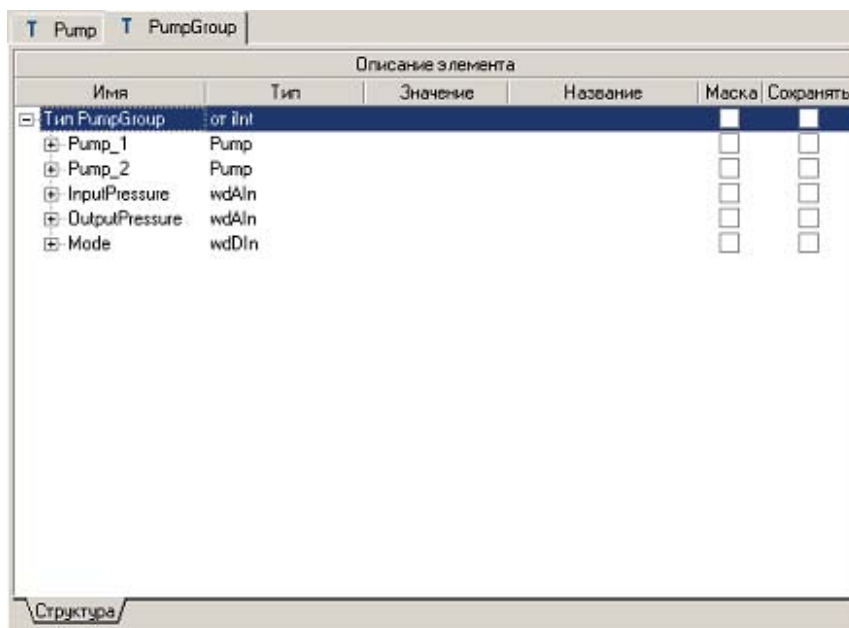


Рис. 7 Тип PumpGroup (группа насосов).

Дадим название всем свойствам.

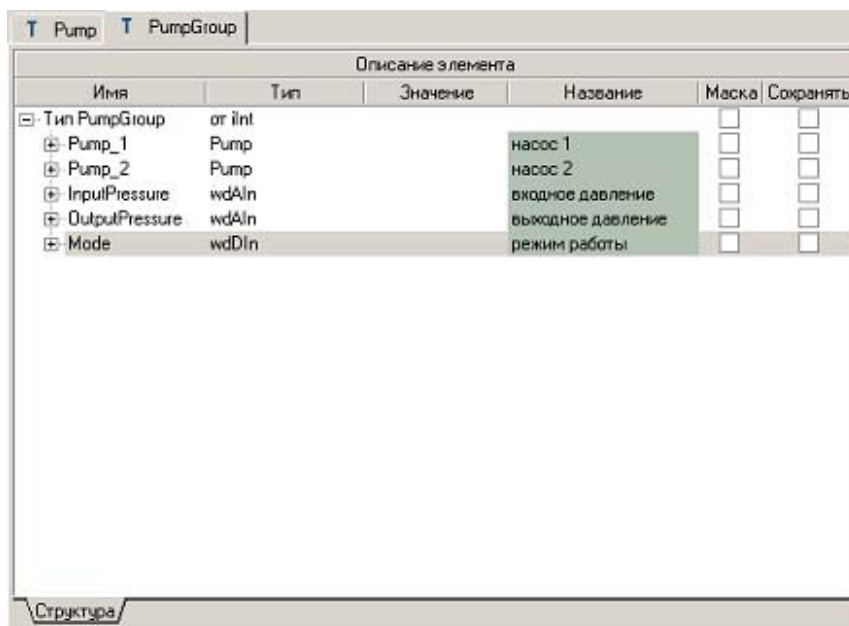


Рис. 8 Тип PumpGroup с названиями элементов.

Сохраните тип PumpGroup.



Рис. 9 Добавление типа PumpGroup в библиотеку.

Следующий шаг: [Создание главного типа \(ЦТП\)](#).

9.4.3.4.5 Создание главного типа (ЦТП)

Создайте новый тип. Назовите его СТР, в качестве базового типа выберите iInt.

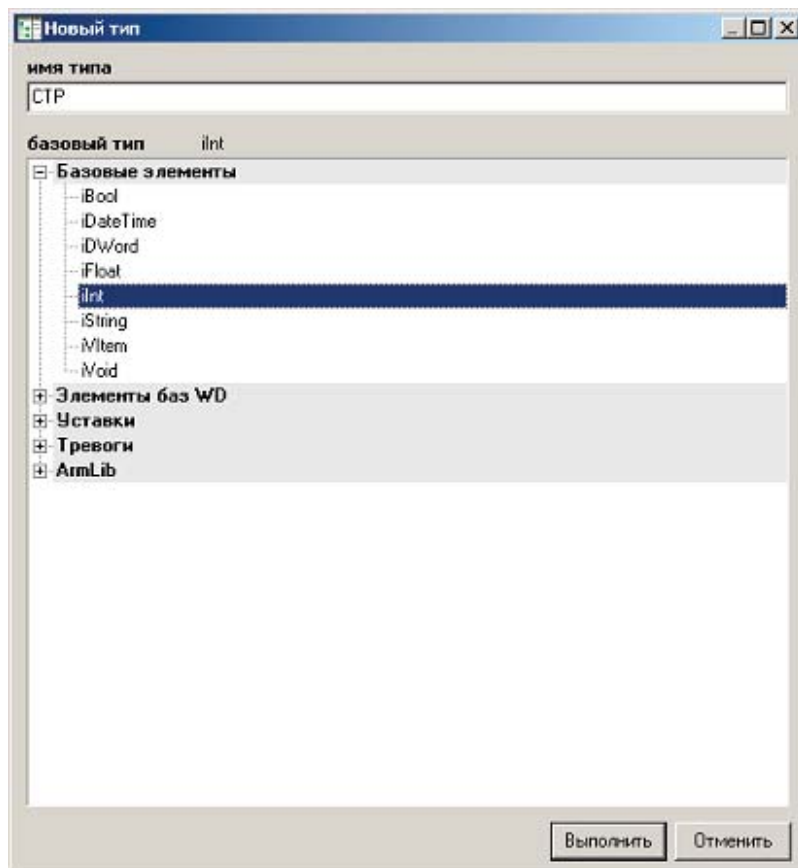


Рис. 1 Создание типа СТР (ЦТП).

Добавьте типу элементы: HotWaterGroup (группа насосов горячей воды) - тип PumpGroup (библиотека "ArmLib"), HeatingGroup (группа насосов отопления) - тип PumpGroup (библиотека "ArmLib"), Fire (пожарная сигнализация) - тип wdDIn (библиотека "Элементы баз WD"), Voltage (напряжение) - тип wdDIn (библиотека "Элементы баз WD").

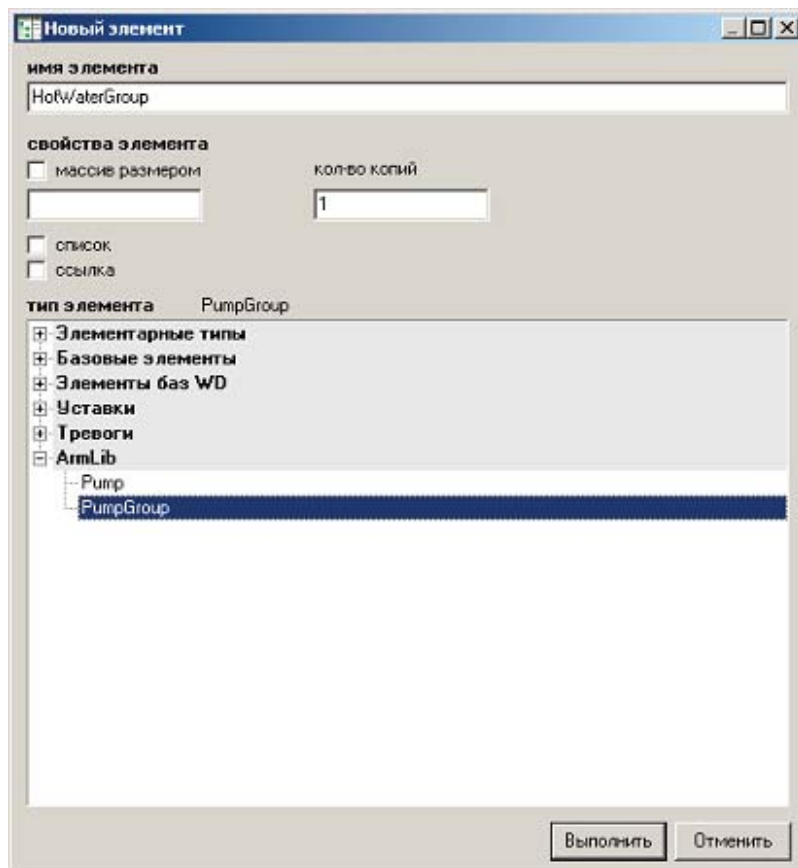


Рис. 2 Добавление свойства HotWaterGroup (группа насосов горячей воды).

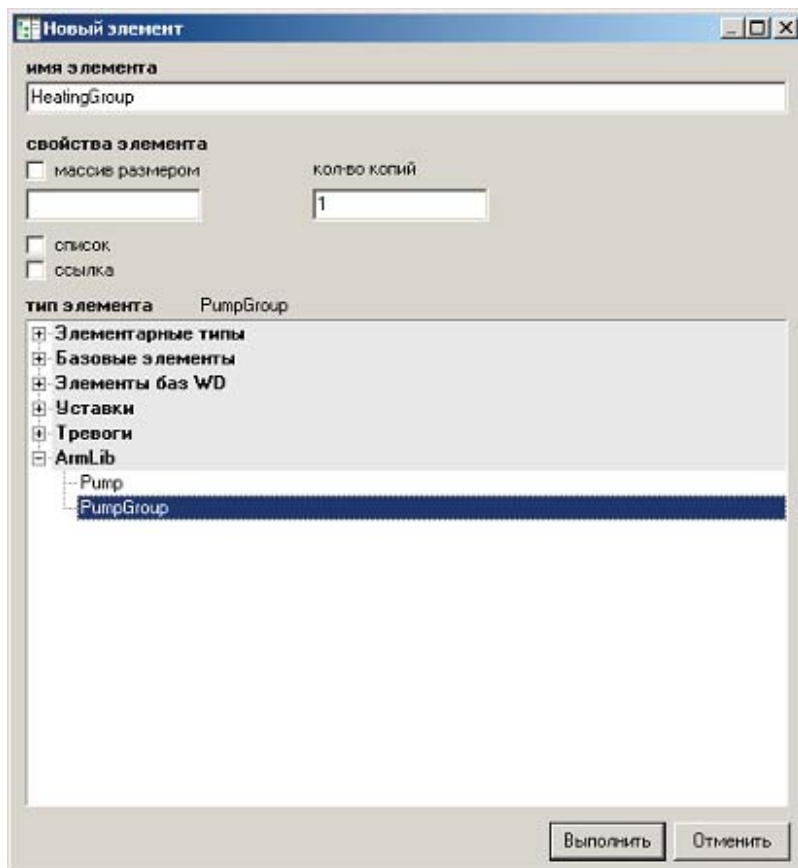


Рис. 3 Добавление свойства HeatingGroup (группа насосов отопления).

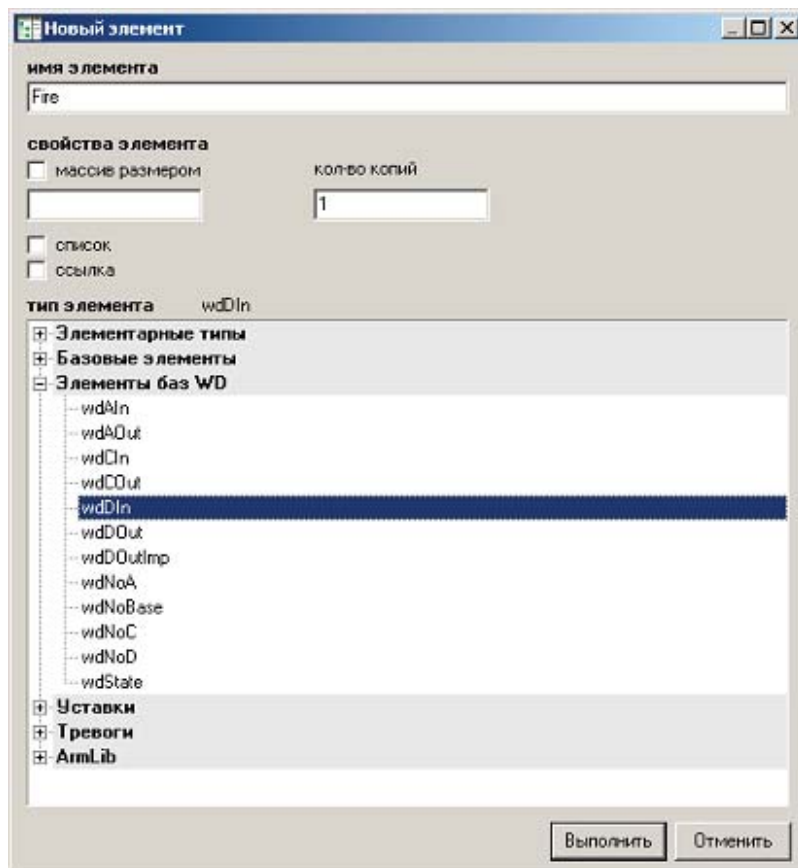


Рис. 4 Добавление свойства Fire (пожарная сигнализация).

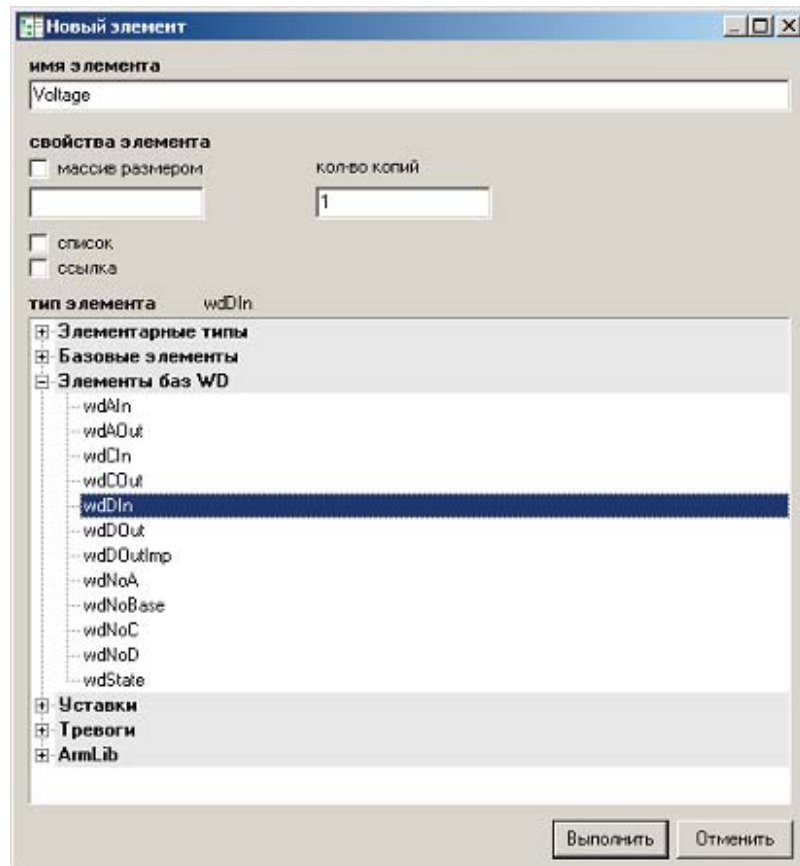


Рис. 5 Добавление свойства Voltage (напряжение).

Окно редактирования типа примет вид.

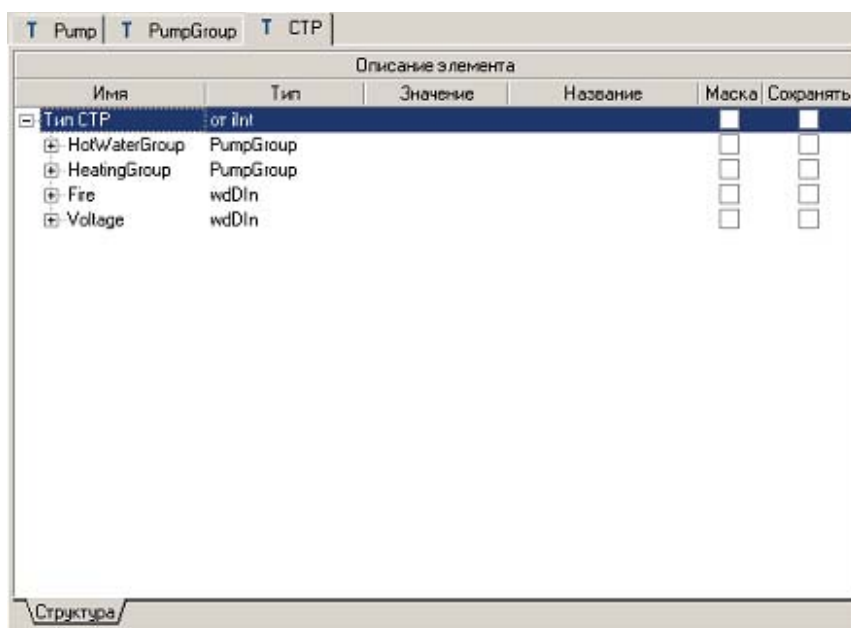


Рис. 6 Тип СТР (ЦТП).

Дадим названия элементам типа.

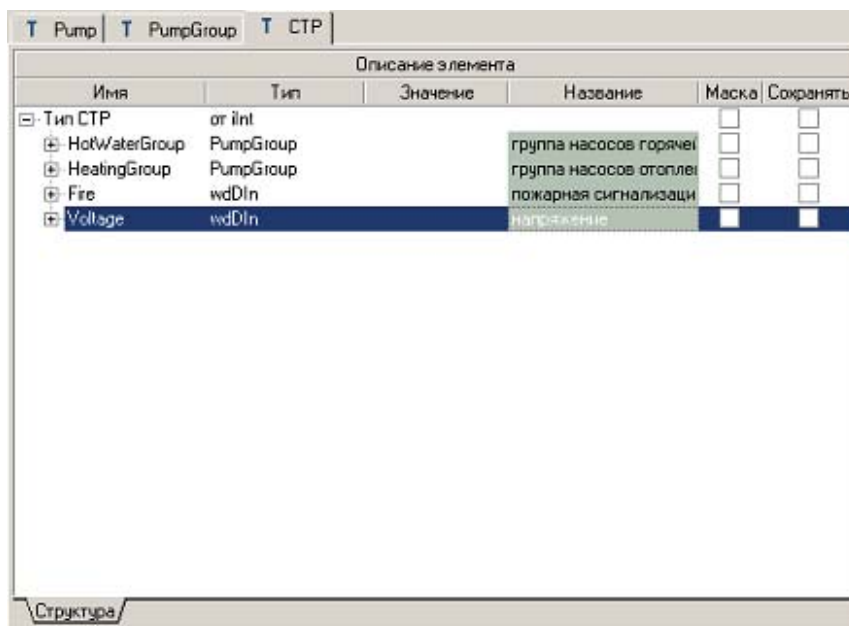



Рис. 7 Тип СТР с названиями элементов.

Сохраните тип СТР.



Рис. 8 Добавление типа СТР (ЦТП) в библиотеку.

Мы закончили создавать типы нашей модели. Сохраните все сделанные изменения ().

Следующий шаг: [Установка связи модели с базами WinDecont.](#)

9.4.3.4.6 Установка связи модели с базами WinDecont

После того как все типы созданы и структурированы, необходимо установить связь элементов модели с базами параметров WinDecont. Связь задается для элементов из библиотеки "Элементы баз WD": wdDIn, wdAIn, wdOutImpl, и т.д. с помощью параметра No (номер).

Пусть сигналы описаны в WinDecont следующим образом.

Входные дискретные

1. Режим работы насосов горячей воды.
2. Состояние насоса горячей воды 1.
3. Состояние насоса горячей воды 2.
4. Режим работы насосов отопления.
5. Состояние насоса отопления 1.
6. Состояние насоса отопления 2.
7. Пожарная сигнализация.
8. Напряжение.

Выходные дискретные

101. Включить насос горячей воды 1.
102. Отключить насос горячей воды 1.
103. Включить насос горячей воды 2.
104. Отключить насос горячей воды 2.
105. Включить насос отопления 1.
106. Отключить насос отопления 1.
107. Включить насос отопления 2.
108. Отключить насос отопления 2.

Входные аналоговые

1. Давление горячей воды на входе.
2. Давление горячей воды на выходе.
3. Давление отопления на входе.
4. Давление отопления на выходе.

В окне редактирования типов откройте закладку с типом СТР (ЦТП).

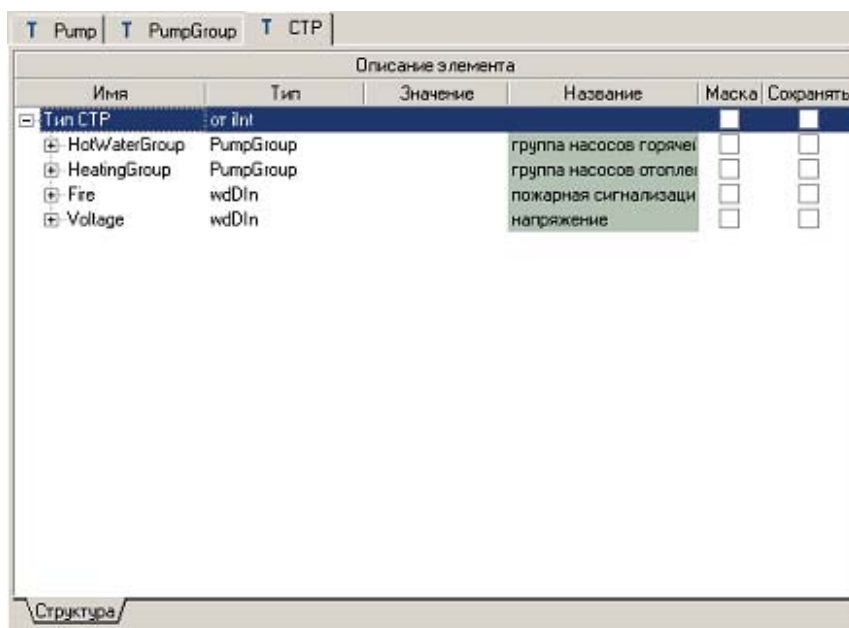


Рис. 1 В окне редактирования типов открыт тип CTP (ЦТП).

Для быстрого доступа к номерам элементов баз WD можно использовать фильтр строк. Перейдите в меню "Просмотр | Фильтр строк".

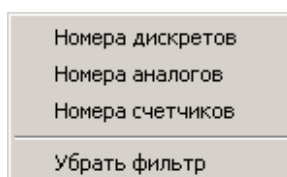


Рис. 2 Меню "Фильтр строк".

Вы можете использовать следующие фильтры:

- **номера дискретов** - показать все поля No (номер) для элементов пронаследованных от типа дискрета (wdDIn, wdOut, wdOutImpl).
- **номера аналогов** - показать все поля No (номер) для элементов пронаследованных от типа аналога (wdAIn, wdAOut).
- **номера счетчиков** - показать все поля No (номер) для элементов пронаследованных от типа счетчика (wdCIn, wdCOut).
- **убрать фильтр** - убрать ранее наложенный фильтр.

Если использовать фильтр, то в окне редактирования типа показываются только элементы привязки к базам WinDecont (номер), что заметно сокращает время редактирования.

Давайте используем фильтр "номера дискретов". Выберите меню "Просмотр | Фильтр строк | Номера дискретов". При этом окно редактирования типов для типа CTP (ЦТП) примет вид.

Описание элемента						
Имя	Тип	Значение	Название	Маска	Сохранять	
Тип СТР	от ilnt			<input type="checkbox"/>	<input type="checkbox"/>	
HotWaterGroup	PumpGroup		группа насосов горячей воды	<input type="checkbox"/>	<input type="checkbox"/>	
Pump_1	Pump		насос 1	<input type="checkbox"/>	<input type="checkbox"/>	
State	wdDIn		состояние	<input type="checkbox"/>	<input type="checkbox"/>	
No	wdNoD	0		<input type="checkbox"/>	<input type="checkbox"/>	
On	wdDOutImp		Включить	<input type="checkbox"/>	<input type="checkbox"/>	
No	wdNoD	0		<input type="checkbox"/>	<input type="checkbox"/>	
Off	wdDOutImp		Выключить	<input type="checkbox"/>	<input type="checkbox"/>	
No	wdNoD	0		<input type="checkbox"/>	<input type="checkbox"/>	
Pump_2	Pump		насос 2	<input type="checkbox"/>	<input type="checkbox"/>	
State	wdDIn		состояние	<input type="checkbox"/>	<input type="checkbox"/>	
No	wdNoD	0		<input type="checkbox"/>	<input type="checkbox"/>	
On	wdDOutImp		Включить	<input type="checkbox"/>	<input type="checkbox"/>	
No	wdNoD	0		<input type="checkbox"/>	<input type="checkbox"/>	
Off	wdDOutImp		Выключить	<input type="checkbox"/>	<input type="checkbox"/>	
No	wdNoD	0		<input type="checkbox"/>	<input type="checkbox"/>	
Mode	wdDIn		режим работы	<input type="checkbox"/>	<input type="checkbox"/>	
No	wdNoD	0		<input type="checkbox"/>	<input type="checkbox"/>	
HeatingGroup	PumpGroup		группа насосов отопления	<input type="checkbox"/>	<input type="checkbox"/>	
Pump_1	Pump		насос 1	<input type="checkbox"/>	<input type="checkbox"/>	
State	wdDIn		состояние	<input type="checkbox"/>	<input type="checkbox"/>	
No	wdNoD	0		<input type="checkbox"/>	<input type="checkbox"/>	
On	wdDOutImp		Включить	<input type="checkbox"/>	<input type="checkbox"/>	
No	wdNoD	0		<input type="checkbox"/>	<input type="checkbox"/>	
Off	wdDOutImp		Выключить	<input type="checkbox"/>	<input type="checkbox"/>	
No	wdNoD	0		<input type="checkbox"/>	<input type="checkbox"/>	
Pump_2	Pump		насос 2	<input type="checkbox"/>	<input type="checkbox"/>	
State	wdDIn		состояние	<input type="checkbox"/>	<input type="checkbox"/>	
No	wdNoD	0		<input type="checkbox"/>	<input type="checkbox"/>	

Рис. 3 Использование фильтра номеров дискретов для типа СТР (ЦТП).

Теперь вам надо набить все номера в соответствии с таблицей сигналов WinDecont приведенной выше. Для этого надо щелкнуть мышкой по полю "Значение" (строка No). Вбить номер (по умолчанию 0) и нажать Enter. Прделаем это на примере состояния первого насоса из группы насосов горячей воды.

Щелкните мышкой по полю "Значение".

Описание элемента						
Имя	Тип	Значение	Название	Маска	Сохранять	
Тип СТР	от ilnt			<input type="checkbox"/>	<input type="checkbox"/>	
HotWaterGroup	PumpGroup		группа насосов горячей воды	<input type="checkbox"/>	<input type="checkbox"/>	
Pump_1	Pump		насос 1	<input type="checkbox"/>	<input type="checkbox"/>	
State	wdDIn		состояние	<input type="checkbox"/>	<input type="checkbox"/>	
No	wdNoD	0		<input type="checkbox"/>	<input type="checkbox"/>	
On	wdDOutImp		Включить	<input type="checkbox"/>	<input type="checkbox"/>	

Рис. 4 Выделение поля "Значение" для редактирования.

Впишите необходимый номер. Состояние первого насоса группы насосов горячей воды имеет второй номер в базе WinDecont.

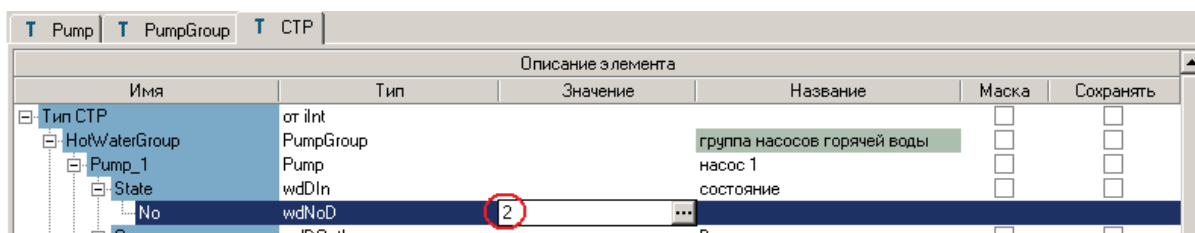


Рис.5 Редактирование привязки элемента к базе WinDecont.

Нажмите Enter. При этом поле станет серого цвета.

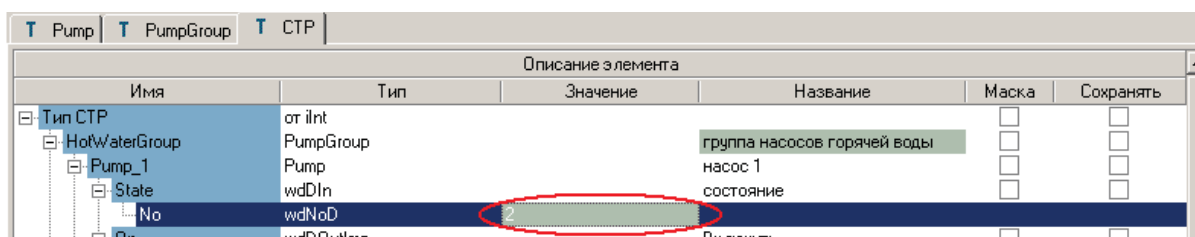


Рис. 6 Поле после редактирования.

Аналогичным образом отредактируйте все привязки.

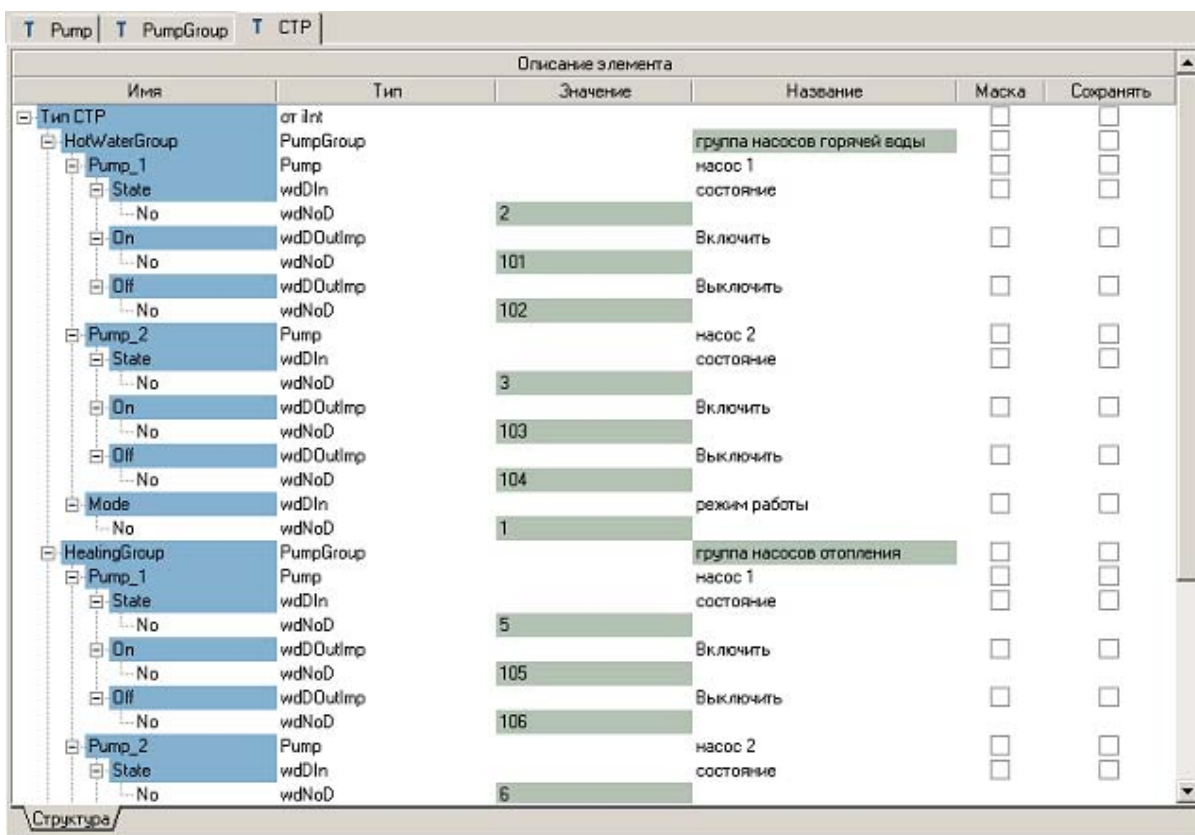


Рис. 7 Тип CTP (ЦТП) после редактирования привязок элементов к базе дискретов WinDecont.

Щелкните мышкой по корню дерева (Тип СТР) и уберите фильтр строк "Просмотр | Фильтр строк | Убрать фильтр".

Прделаем аналогичную операцию для номеров аналогов. Примените фильтр по номерам аналогов "Просмотр | Фильтр строк | Номера аналогов" и отредактируйте все привязки.

Описание элемента						
Имя	Тип	Значение	Название	Маска	Сохранять	
Тип СТР	ot int			<input type="checkbox"/>	<input type="checkbox"/>	
HotWaterGroup	PumpGroup		группа насосов горячей воды	<input type="checkbox"/>	<input type="checkbox"/>	
InputPressure	wdAln		входное давление	<input type="checkbox"/>	<input type="checkbox"/>	
No	wdNoA	1				
OutputPressure	wdAln		выходное давление	<input type="checkbox"/>	<input type="checkbox"/>	
No	wdNoA	2				
HeatingGroup	PumpGroup		группа насосов отопления	<input type="checkbox"/>	<input type="checkbox"/>	
InputPressure	wdAln		входное давление	<input type="checkbox"/>	<input type="checkbox"/>	
No	wdNoA	3				
OutputPressure	wdAln		выходное давление	<input type="checkbox"/>	<input type="checkbox"/>	
No	wdNoA	4				

Рис. 8 Тип СТР (ЦТП) после редактирования привязок элементов к базе аналогов WinDecont.

Щелкните мышкой по корню дерева (Тип СТР) и уберите фильтр строк "Просмотр | Фильтр строк | Убрать фильтр".

Следующий шаг: [Установки проекта.](#)

9.4.3.4.7 Установки проекта

После того, как созданы все типы модели, необходимо произвести установки проекта.

1. Указать главный тип модели.
2. Архивное хранилище (*).
3. Такт модели.

* - необязательно.

Главный тип нашей модели это СТР. Такт модели характеризует, с какой частотой запускается тактовая функция главного типа (частота обновления данных в модели). Зададим такт равный 500 мс.

Щелкните мышкой по значку  (установки проекта). Укажите в диалоге главный тип и такт модели.

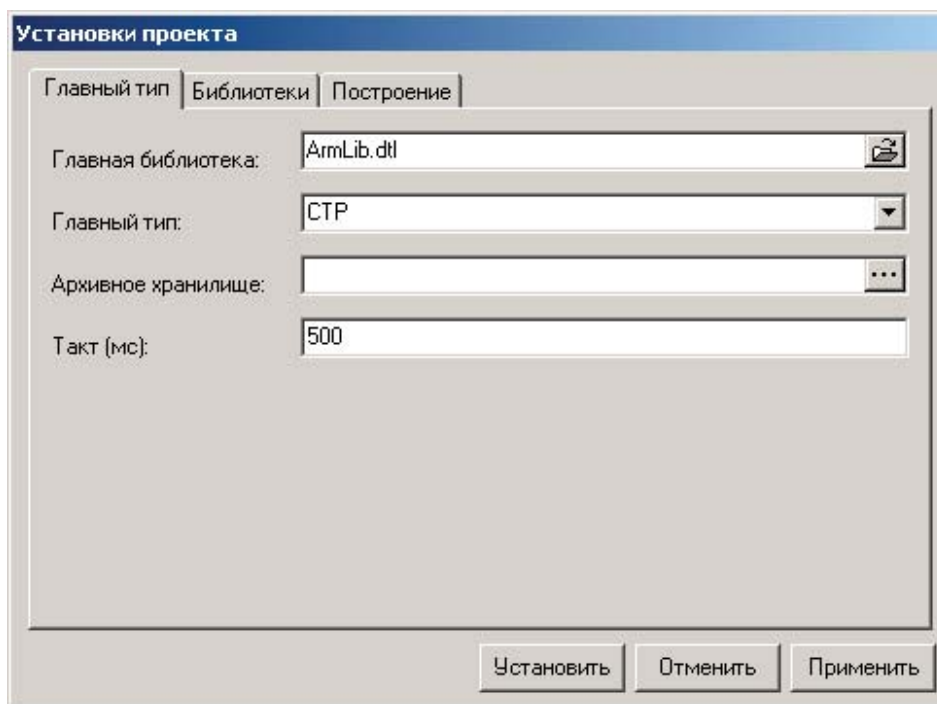


Рис. 1 Установки проекта.

Нажмите "Установить".

Следующий шаг: [Завершение создания модели](#).

9.4.3.4.8 Завершение создания модели

Нам осталось только построить и запустить нашу модель.

Постройте модель  (предварительно сохранив все изменения .

!!!Не забывайте, что в библиотеке модели типы, которые используются в других типах, должны идти раньше последних !!!

Если все было сделано правильно, в окне сообщений будет написано, что проект собран и ошибок не найдено.

```
18:09:54: Генерация кода C:\alex\works\Manuals\ModelBuilder\Models\ARM\RESULT\ArmLib.h
18:09:54: Генерация кода C:\alex\works\Manuals\ModelBuilder\Models\ARM\RESULT\ArmLib.cpp
18:09:54: Генерация кода C:\alex\works\Manuals\ModelBuilder\Models\ARM\RESULT>Main.h
18:09:54: Генерация кода C:\alex\works\Manuals\ModelBuilder\Models\ARM\RESULT>Main.cpp
18:09:54: Построение MAKE-файла C:\alex\works\Manuals\MODELB-1\Models\ARM\RESULT\tespcap.mak
18:09:55: Выполнение MAKE-файла ...
MAKE Version 5.2 Copyright (c) 1987, 2000 Borland
      D:\PROCRA-1\Borland\CBUILD-2\BIN\bcc32.exe -MD -O2 -b -k- -vi -H=dnl.csa -Vx -Ve -X- -a8 -tMD -tWM -c -D_DEBUG;NO_STRICT -ID:\PROCRA-1\
Borland C++ 5.6 for Win32 Copyright (c) 1993, 2002 Borland
C:\alex\works\Manuals\MODELB-1\Models\ARM\RESULT\ArmLib.cpp:
C:\alex\works\Manuals\MODELB-1\Models\ARM\RESULT>Main.cpp:
Loaded pre-compiled headers.

18:10:02: Выполнение MAKE-файла завершено
18:10:02: Построение LINK-файла
18:10:02: Сборка проекта ...
Turbo Incremental Link 5.60 Copyright (c) 1997-2002 Borland

18:10:04: Сборка проекта завершена
```

Добавьте модель в список моделей WinDecont. Для этого откройте программу WinDecont.

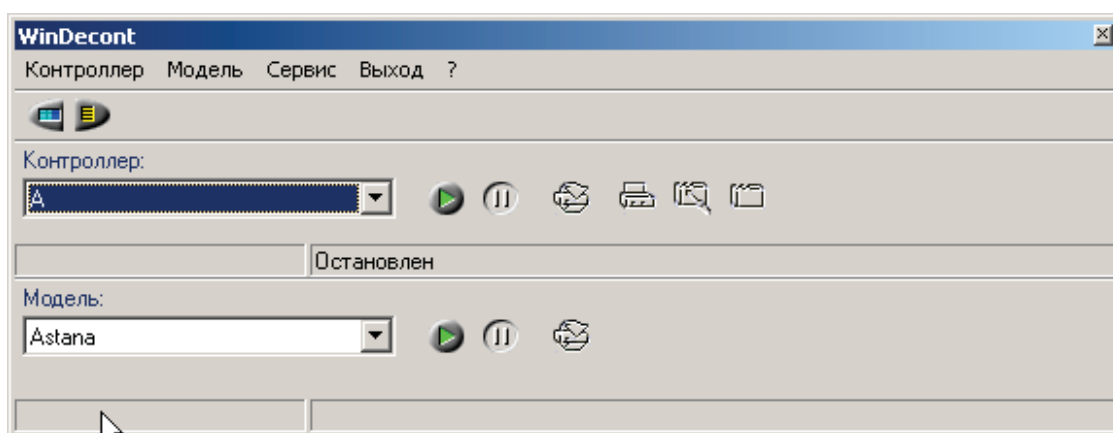


Рис. 1 WinDecont.

Откройте окно параметров WinDecont (меню Сервис | Параметры).

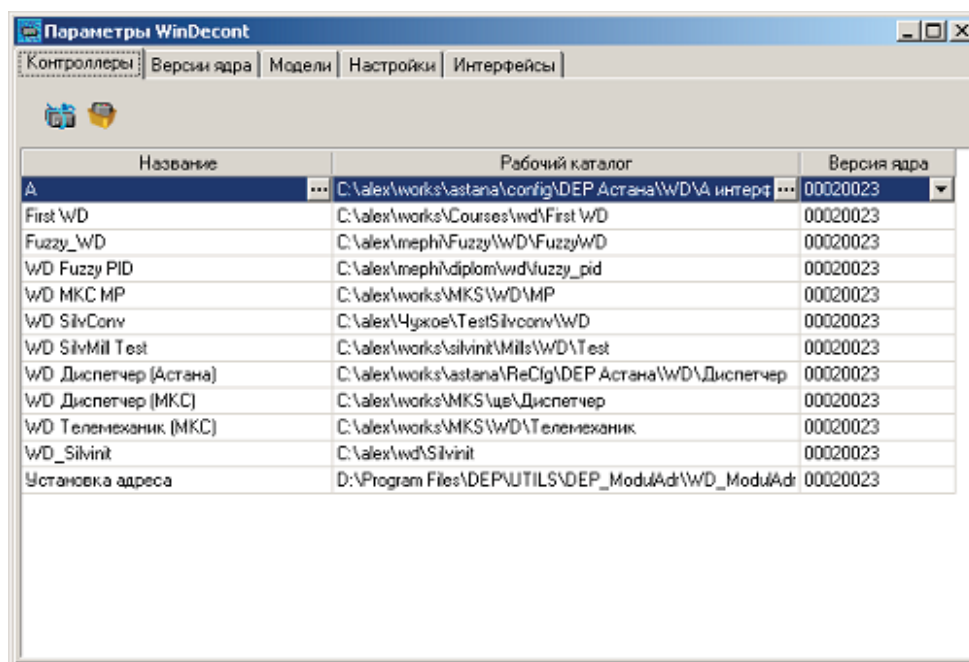


Рис. 2 Параметры WinDecont.

Откройте закладку "Модели", щелкните мышкой по значку  (установить новую модель). Напишите название модели (APM) и укажите путь к файлу Model.dll (... | ваша модель | RESULT).

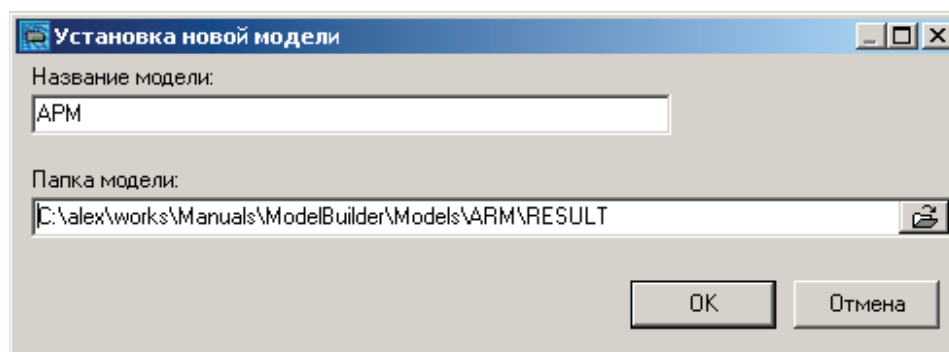


Рис. 3 Установка новой модели.

Закройте окно параметров WinDecont. Выберите модель APM из списка и нажмите  (запустить модель).

Все! Модель создана и запущена. Пора заняться отображением.

Следующий шаг: [Создание отображения.](#)

9.4.3.5 Создание отображения

9.4.3.5.1 Создание нового проекта

Сначала создадим новый проект. Запустите C++Builder и откройте новый depOPCProject. Для этого перейдите в меню "File | New | Other". Откройте закладку DEP, выберите depOPCProject и нажмите "OK".

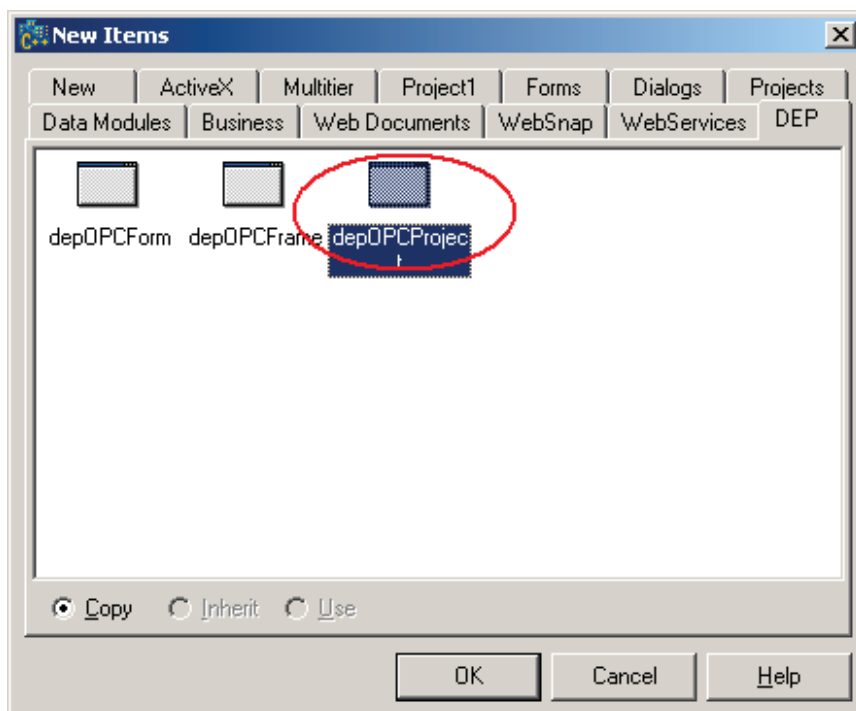


Рис. 1 Создание нового проекта depOPCProject.

При этом на экране появится новая форма с объектом depOPCDesigner на ней. Сохраните проект "File | Save All". Назовите форму ArmUnit, а проект ArmView.

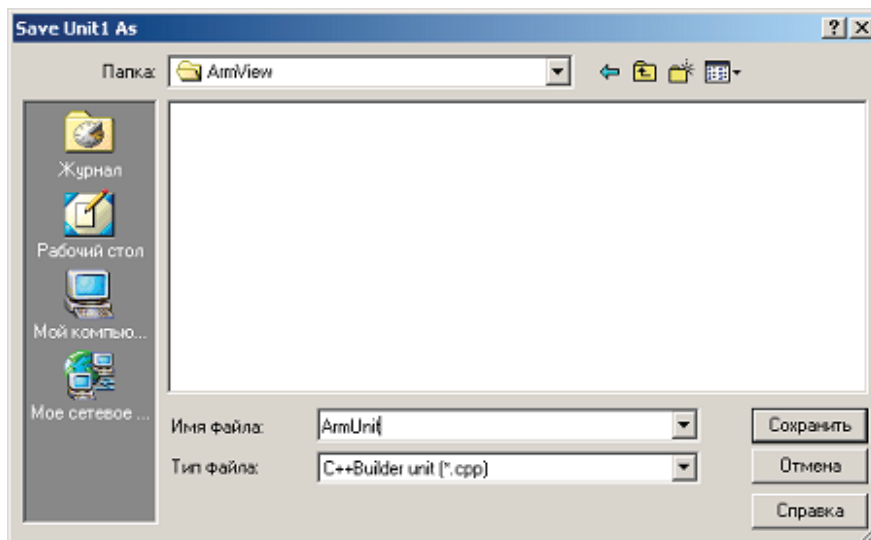


Рис. 2 Сохранение файла формы.

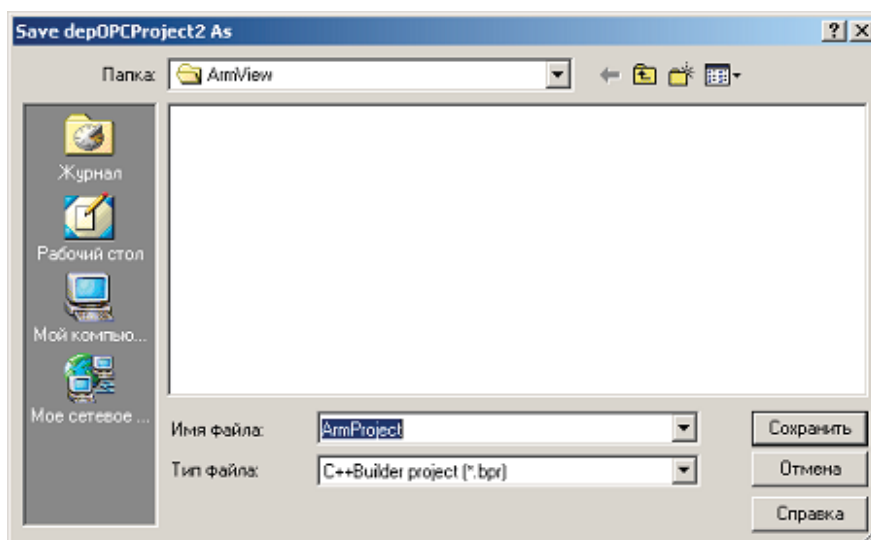


Рис. 3 Сохранение проекта.

Пора заняться самим отображением. Давайте подумаем, какие компоненты нам понадобятся для рисования нашего технологического экрана.

Следующий шаг: [Разбиение изображения на компоненты.](#)

9.4.3.5.2 Разбиение изображения на компоненты

Подумаем, исходя из внешнего вида технологического экрана, какие компоненты нам понадобятся.

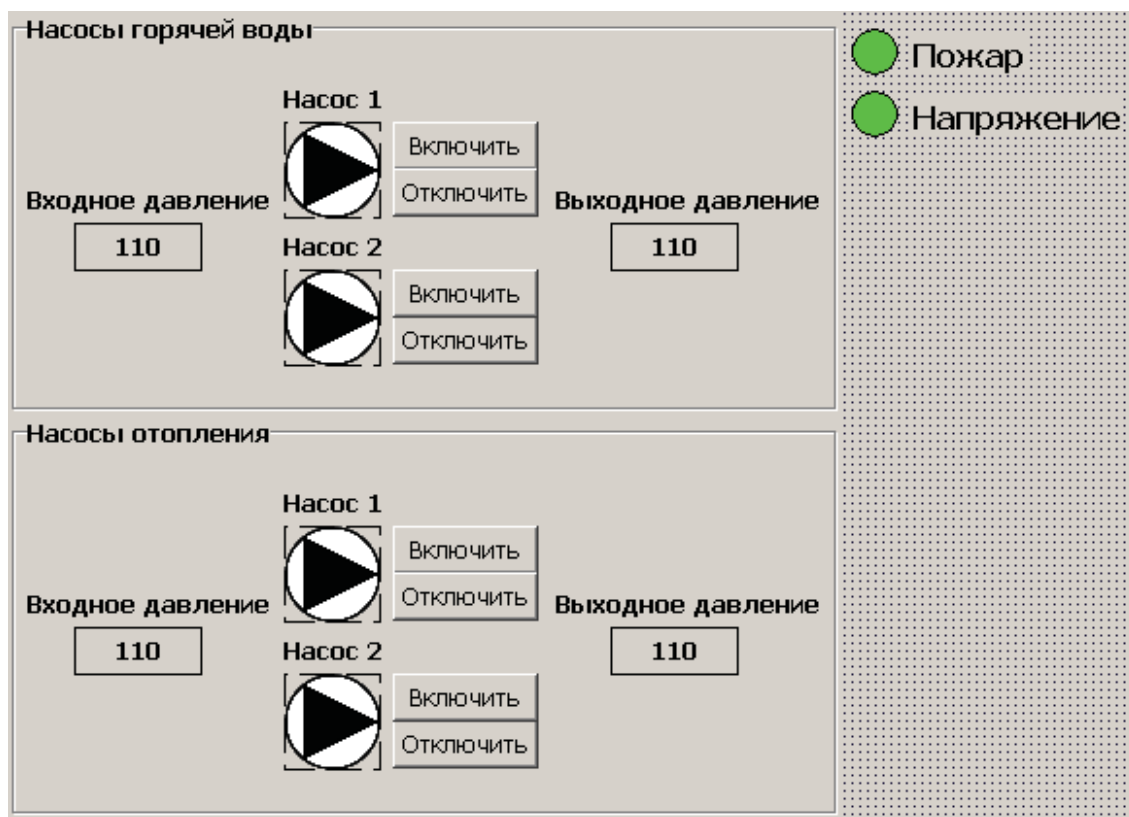


Рис. 1 Вид технологического экрана.

Для разделения групп насосов на экраны будем использовать GroupBox (закладка "Standard"). Для отображения количественных сигналов (входное, выходное давления) логично использовать метки. Будем использовать DEPLabel (закладка "DEP"). Для рисования насосов, пожарной сигнализации и напряжения будем использовать элементарные формы (круг, квадрат, треугольник) - DEPShape (закладка "DEP"). Для информационных надписей будем использовать обычные метки - TLabel (закладка "Standard").

Используя эти компоненты, повторите (примерно) вид технологического экрана.

Сохраните все изменения "File | Save All".

Следующий шаг: [Привязка отображения к модели.](#)

9.4.3.5.3 Привязка отображения к модели

9.4.3.5.3.1 Введение

После того как отображение нарисовано на форме с помощью компонентов C++Builder, настает пора его оживить. Для этого мы должны привязать свойства компонентов, отвечающие за оживление, к элементам модели.

Перед тем как начинать привязку, запустите в WinDecont нашу модель (APM).

Привяжем сначала метки, отвечающие за вывод информации о входных и выходных давлениях.

Следующий шаг: [Привязка меток](#).

9.4.3.5.3.2 Привязка меток

Начнем привязку с метки, отвечающей за отображение значения входного давления для насосов горячей воды. Щелкните правой кнопкой мышки по метке и выберите из выпадающего меню "Редактор OPC состояний".

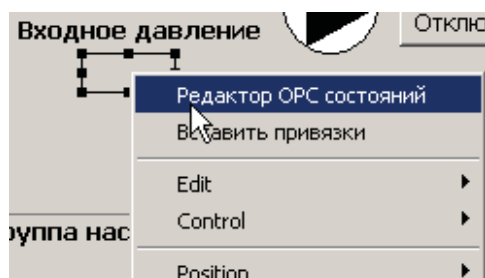


Рис. 1 Открытие редактора OPC состояний.

В результате откроется редактор OPC состояний ("OPC-дизайнер"). Синим цветом, в дереве объекта формы, выделена наша метка.

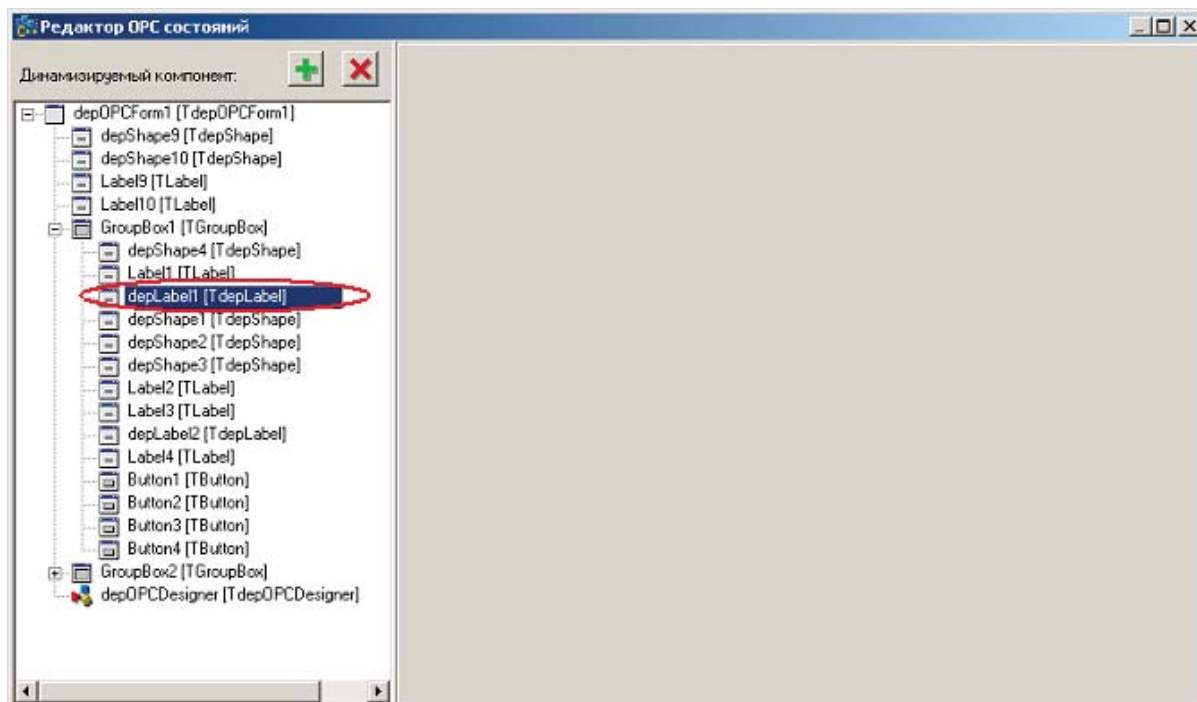


Рис. 2 Редактор OPC состояний.

Нажмите "Добавить привязку" . Выберите свойство метки, к которому хотите сделать привязку. В нашем случае это Caption - текст метки. Нажмите "ОК".

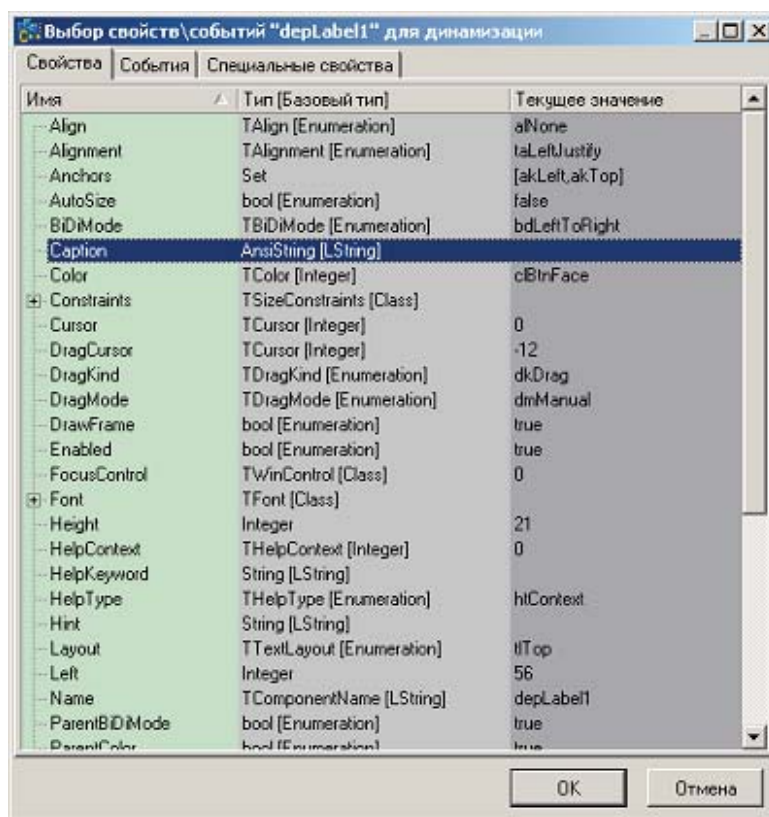



Рис. 3 Выбор свойства метки для привязки.

Выберите элемент модели для привязки . В нашем случае мы хотим привязать текст метки к входному давлению для группы насосов горячей воды, поэтому мы должны привязаться к элементу "HotWaterGroup.InputPressure". Выберите элемент и нажмите "Выбрать".

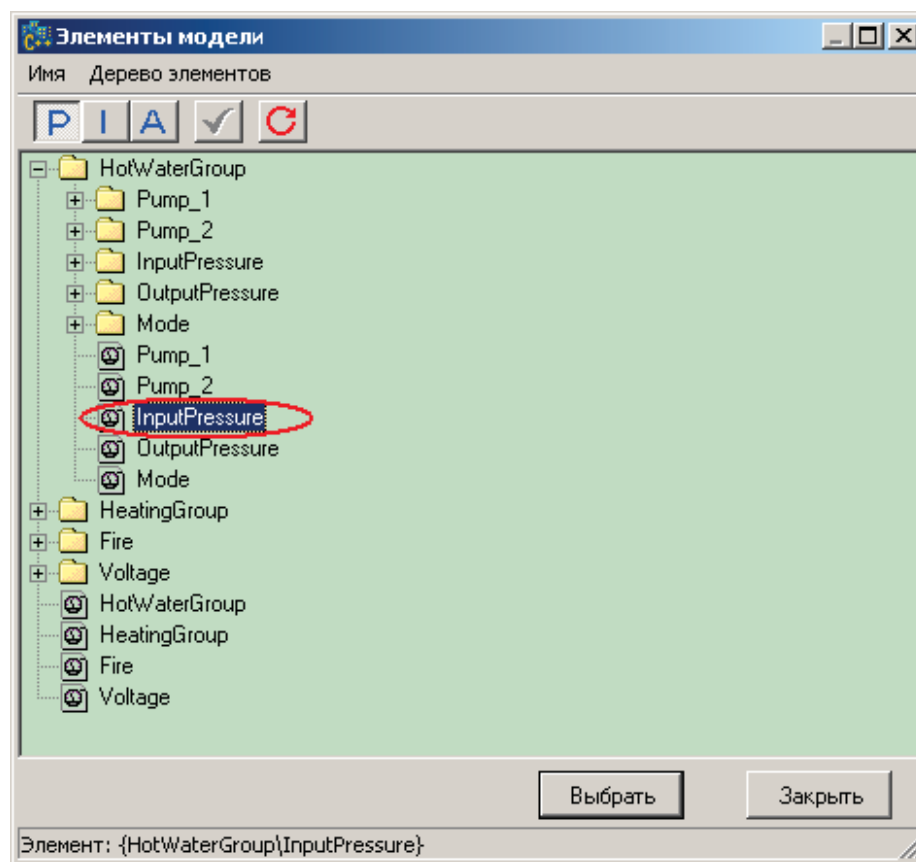


Рис. 4 Выбор элемента для привязки.

В поле "Caption" в окне редактирования состояний напишите %f.

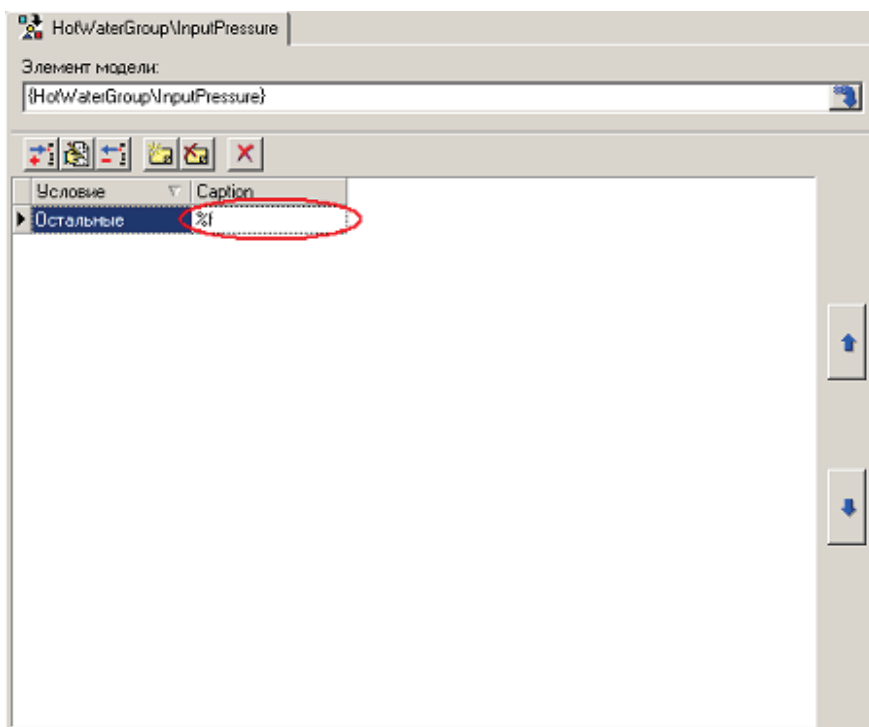


Рис. 5 Редактирование поля Caption для метки входного давления насосов горячей воды.

Здесь следует дать пояснение. При выводе численной (строковой) информации мы должны взять значение OPC-элемента и записать его в метку (свойство Caption). При этом используются, так называемые, маски. С помощью маски задается тип значения, который мы хотим выводить в текстовом виде.

Маски

- **%f** - числа с плавающей запятой.
- **%d** - целые числа.
- **%s** - строки.


Так как входное давление это аналог, то тип данных у нас float (числа с плавающей запятой), поэтому мы используем маску %f.


Условия для состояния OPC-элемента просматриваются сверху вниз, и последним всегда идет "Остальные", которое используется, если не подошло ни одно из предыдущих условий.

Исходя из выше сказанного, привязка работает следующим образом: берется значение элемента модели, оно рассматривается как число с плавающей запятой, и записывается в метку.

На это привязка метки закончилась. Теперь наша метка будет отображать значение входного давления для группы насосов горячей воды. Закройте редактор OPC состояний.

Привяжем метку выходного давления для группы насосов.

Щелкните правой кнопкой мышки по метке и выберите "Редактор OPC состояний". Нажмите добавить привязку  и

выберите свойство Caption. Выберите элемент модели для привязки . В нашем случае мы хотим привязать текст метки к выходному давлению для группы насосов горячей воды, поэтому мы должны привязаться к элементу "HotWaterGroup.OutputPressure". Выберите элемент и нажмите "Выбрать".

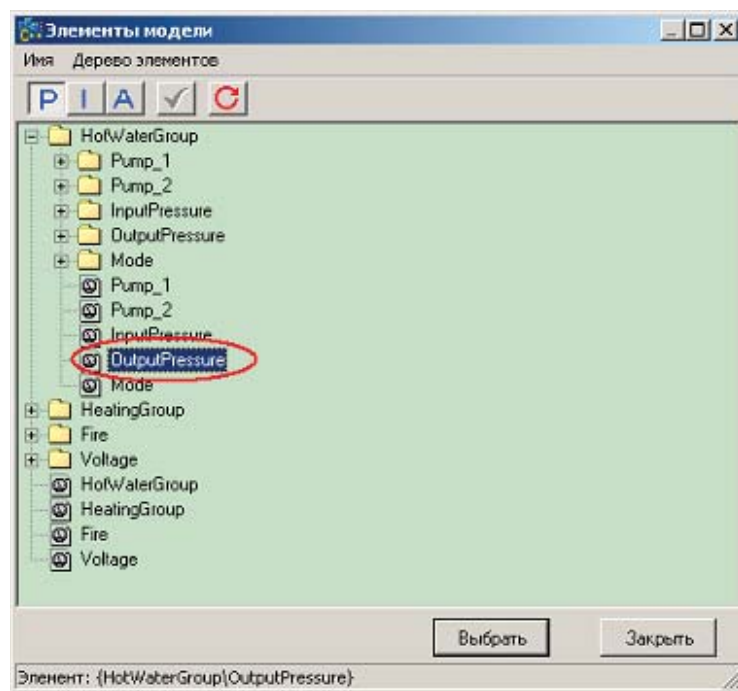


Рис. 6 Выбор элемента для привязки.

В поле Caption в окне редактирования состояний напишите %f.

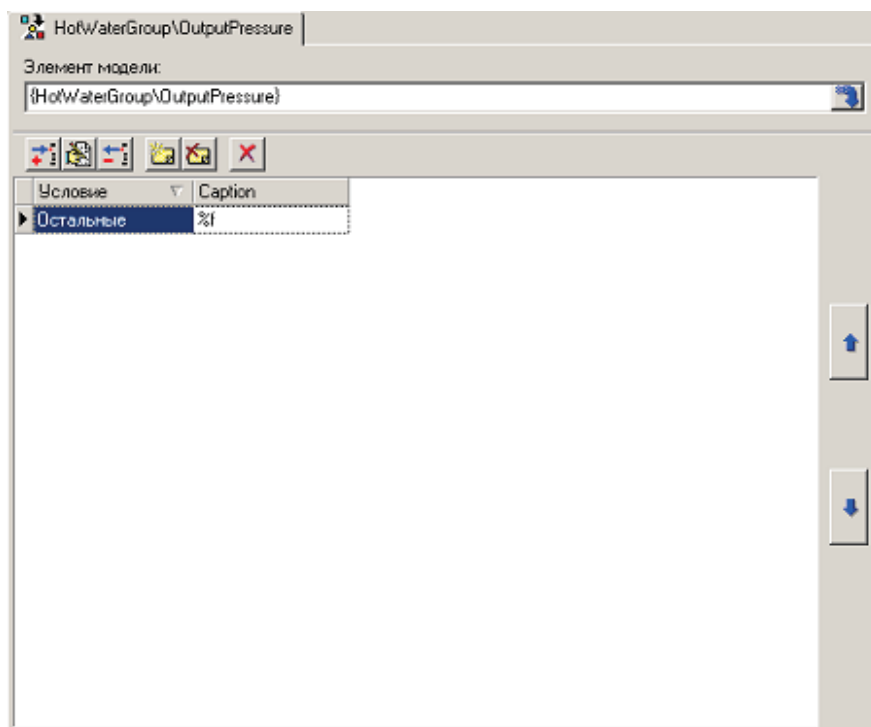


Рис. Редактирование поля Caption для метки выходного давления насосов горячей воды.


Привяжите аналогичным образом метки входного и выходного давлений для группы насосов отопления, к элементам HeatingGroup.InputPressure и HeatingGroup.OutputPressure соответственно.

Теперь привяжем объекты DEPSHare, отвечающие за индикацию состояния насосов.

Следующий шаг: [Привязка изображений насосов.](#)

9.4.3.5.3.3 Привязка изображений насосов

Как вы помните, в соответствии с заданием мы должны менять цвет фона изображений насосов в зависимости от их состояния. Значит, мы должны привязать свойство, отвечающее за цвет фона к элементу модели, отвечающему за состояние насоса.

Щелкните правой кнопкой мышки по окружности насоса 1 из группы насосов горячей воды и выберите, из выпадающего меню, пункт "Редактор OPC состояний". Добавьте привязку . Выберите свойство Brush.Color (цвет фона). Нажмите "ОК".

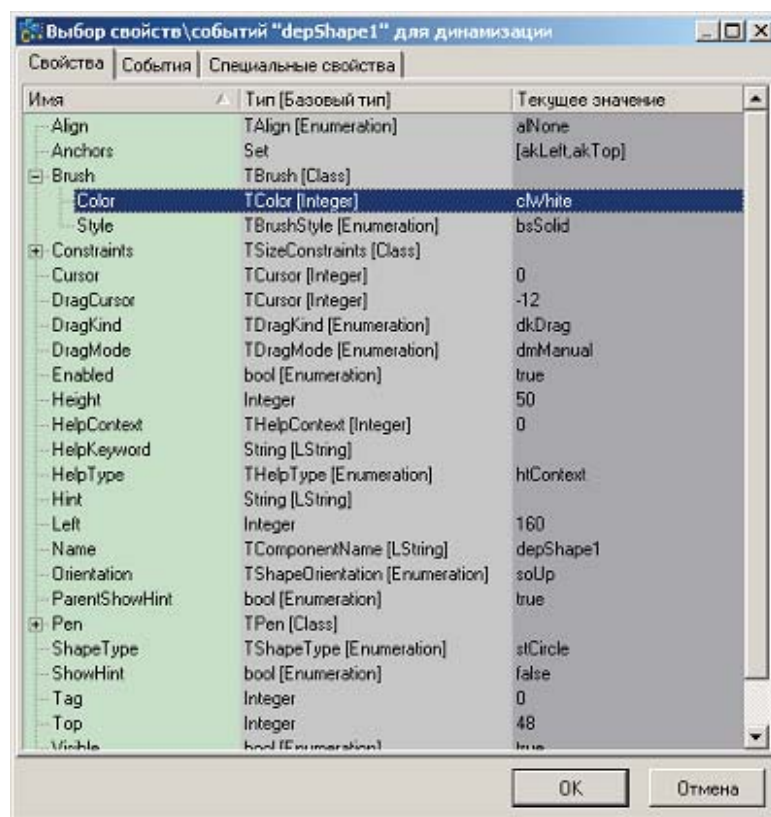



Рис. 1 Выбор свойства Brush.Color.

Нажмите "Выбрать элемент модели" . Выберите элемент HotWaterGroup.Pump_1.State (состояние первого насоса из группы насосов горячей воды). Нажмите "Выбрать".

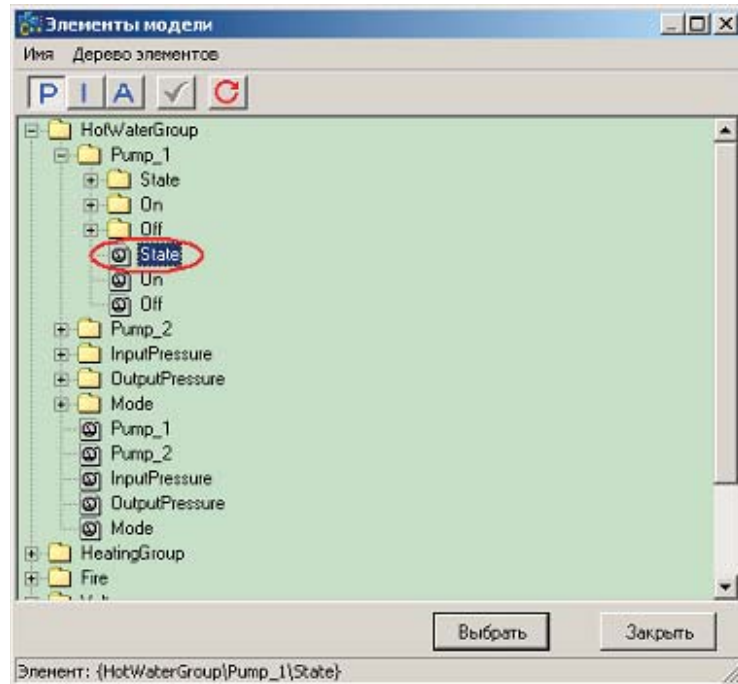



Рис. 2 Выбор элемента модели, отвечающего за состояние первого насоса горячей воды.

Добавьте включенное, выключенное и неопределенное состояние. Для этого щелкните мышкой по значку  (добавить состояние). Определимся, чем характеризуется каждое состояние. Насос включен - value (значение дискрета) == 1, насос выключен - value (значение дискрета) == 0, состояние неопределенно - Quality. Неопределенность (неопределенное значение дискрета). Добавьте в соответствии с этим состояния для насоса. Для добавления каждого состояния щелкаете мышкой по значку "Добавить состояние", выбираете нужное состояние, после чего нажимаете "OK".

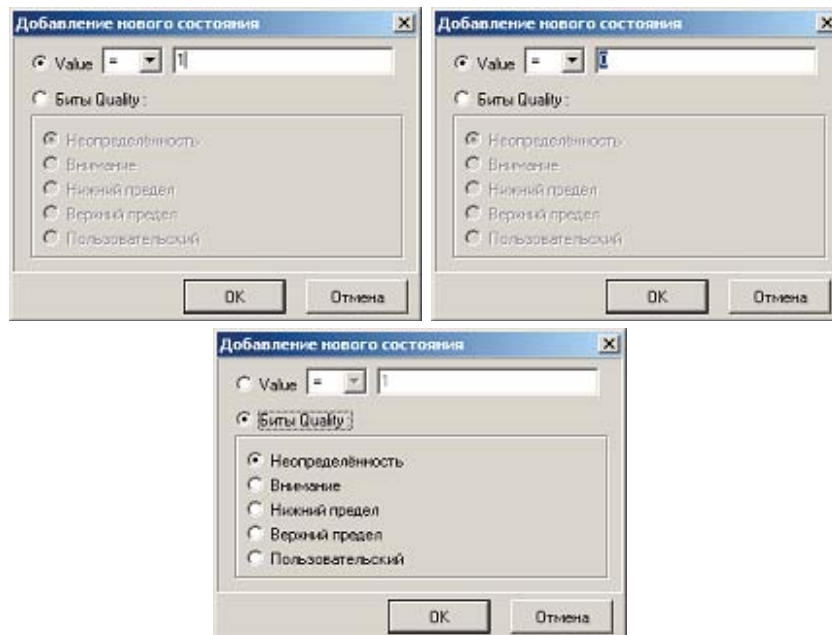


Рис. 3 Добавление включенного, выключенного и неопределенного состояний насоса.

Выберите для каждого состояния цвет в соответствии с заданием на АРМ. Для этого щелкните мышкой по полю Brush.Color напротив каждого состояния и выберите цвет из списка.

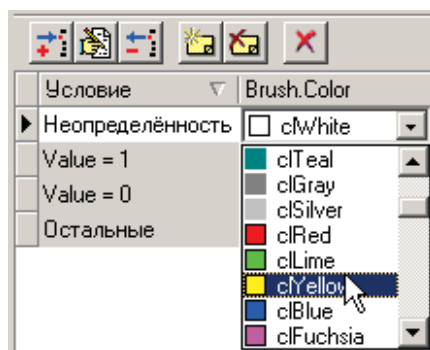


Рис. 4 Выбор цвета для неопределенного состояния.

После выбора цвета для каждого состояния окно редактирования состояний примет вид.

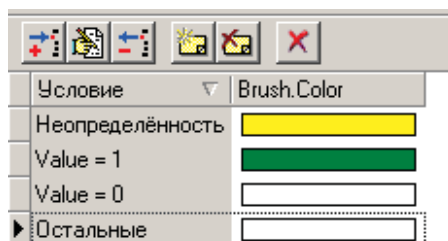



Рис. 5 Окно редактирования состояний после окончания редактирования.

Привяжите аналогичным образом остальные насосы к элементам модели. Для второго насоса горячей воды это HotWaterGroup.Pump_2.State, для первого насоса отопления - HeatingGroup.Pump_1.State, для второго насоса отопления - HeatingGroup.Pump_2.State.

Теперь привяжем кнопки, отвечающие за управление насосами.

Следующий шаг: [Привязка кнопок](#).

9.4.3.5.3.4 Привязка кнопок

Щелкните правой кнопкой мышки по кнопке "Включить" для первого насоса отопления. Выберите, из выпадающего меню, "Редактор OPC состояний". Нажмите добавить привязку . Откройте закладку "События".

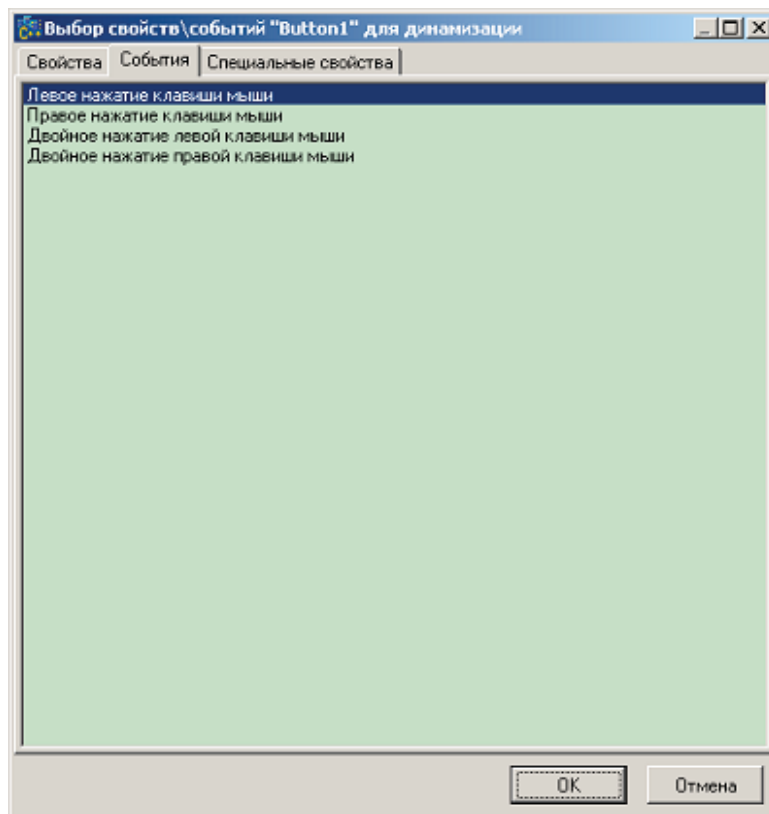


Рис. 1 Закладка "События" окна выбора свойств.

Выберите необходимое событие, в нашем случае это "Левое нажатие клавиши мыши", и нажмите "ОК". Окно редактирования привязок для кнопки примет вид.

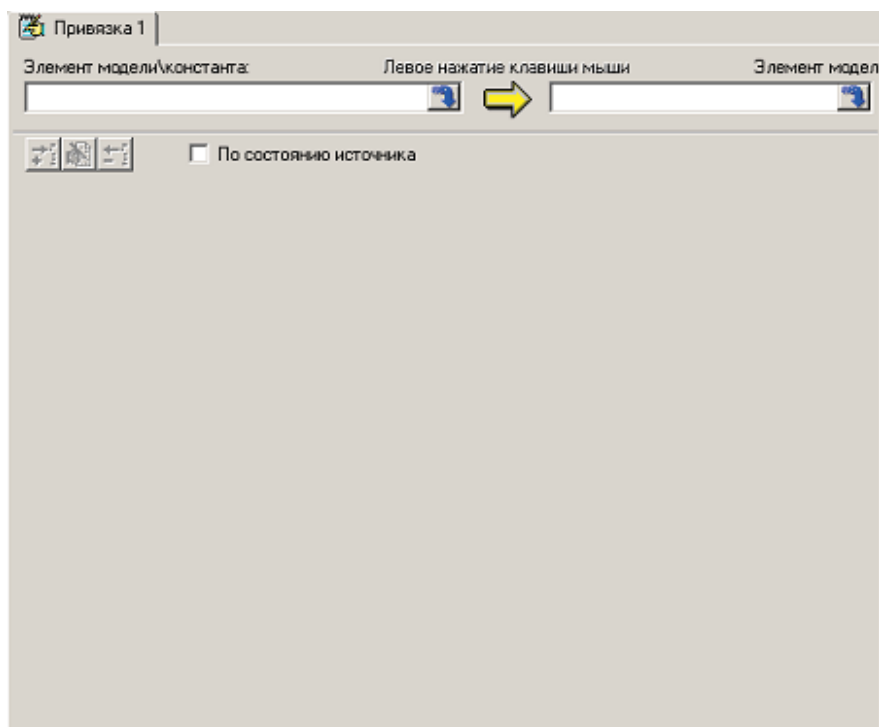


Рис. 2 Окно редактирования привязки для привязки типа - событие.

В левом поле надо написать константу для записи, в нашем случае 1, а в правом поле выбрать элемент модели, куда будет производиться запись, в нашем случае HotWaterGroup.Pump_1.On.

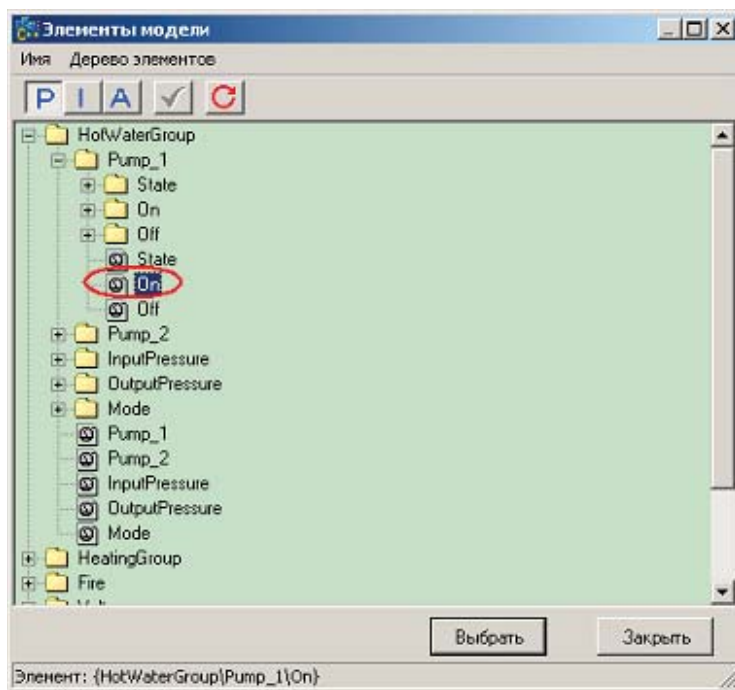


Рис. 3 Выбор элемента включения первого насоса горячей воды.

После этого окно редактирования привязок примет вид.

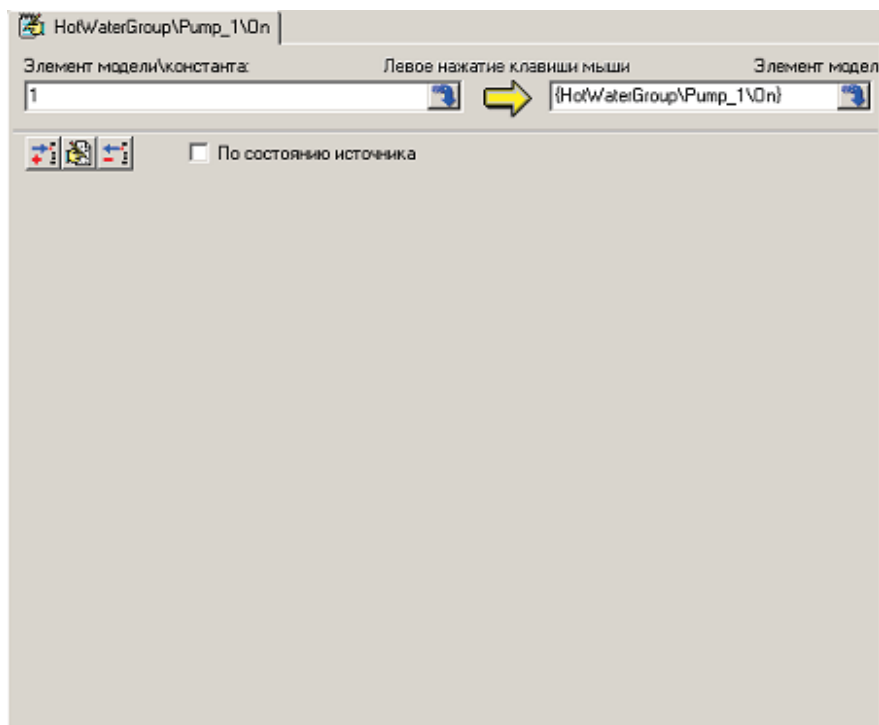


Рис. 4 Окно редактирования привязок, для привязки типа - событие, после окончания редактирования.


Привяжите аналогичным способом остальные кнопки. Кнопку выключения первого насоса горячей воды к элементу HotWaterGroup.Pump_1.Off, включение второго насоса горячей воды - HotWaterGroup.Pump_2.On, выключение второго насоса горячей воды - HotWaterGroup.Pump_2.Off, включения первого насоса отопления - HeatingGroup.Pump_1.On, выключения первого насоса отопления - HeatingGroup.Pump_1.Off, включения второго насоса отопления - HeatingGroup.Pump_2.On, выключения второго насоса отопления - HeatingGroup.Pump_2.Off.

И последнее, привяжем индикаторы, отвечающие за пожарную сигнализацию и напряжение.

Следующий шаг: [Привязка индикаторов пожара и напряжения.](#)

9.4.3.5.3.5 Привязка индикаторов пожара и напряжения

Привязка индикаторов напряжения и пожара производится аналогично привязки изображений насосов.

Щелкните правой кнопкой мышки по индикатору пожара, в выпадающем меню выберите "Редактор OPC состояний". В окне свойств выберите свойство Brush.Color, нажмите "ОК". Нажмите "Выбрать элемент модели" . Выберите элемент Fire, нажмите "Выбрать".

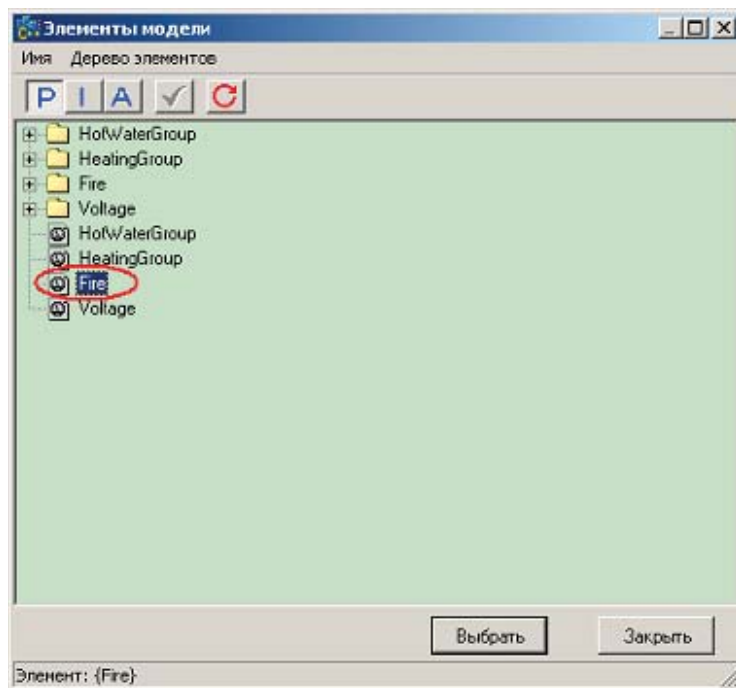


Рис. 1 Выбор элемента, отвечающего за пожарную сигнализацию.

Добавьте состояния 1 - сработала пожарная сигнализация, 0 - пожара нет, неопределенно - значение датчика неопределенно.

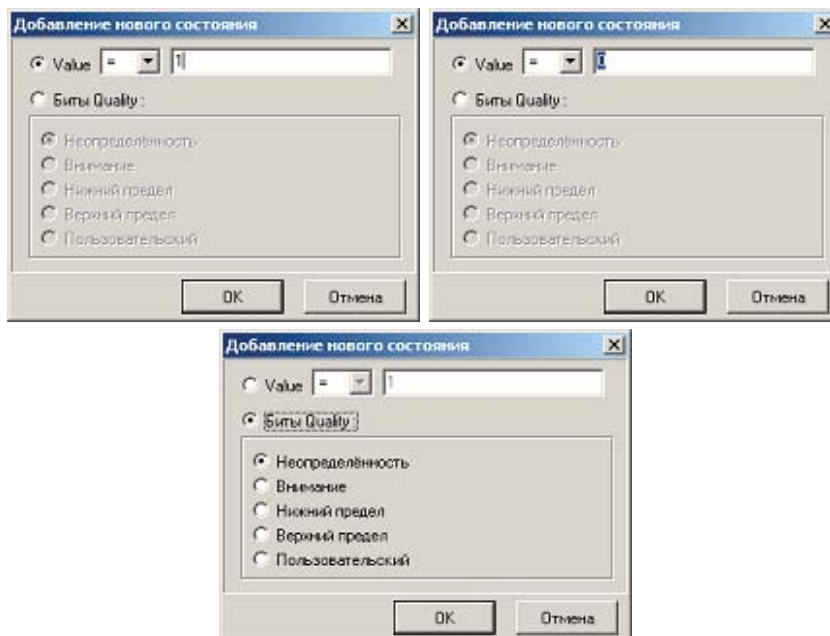


Рис. 2 Добавление состояний для индикатора пожара.

Выберите цвет для каждого состояния в соответствии с заданием на АРМ.

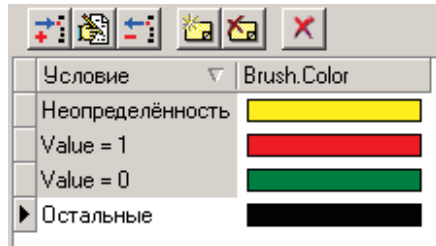


Рис. 3 Выбор цветов для состояний индикатора.

Аналогичным образом привязываем индикатор напряжения к элементу Voltage.

На этом привязка окончена. Сохраните все изменения (File | Save All).

Следующий шаг: [Завершение создания отображения.](#)

9.4.3.5.3.6 Завершение создания отображения

Нам осталось только собрать и запустить отображение.

Соберите проект Ctrl + F9. Если вы все сделали правильно, ошибок при компилировании не будет. Запустите отображение Run | Run и посмотрите, что у вас получилось.



Рис. 1 Работающее отображение.

Не удивляйтесь, что у вас все картинки желтые. Это связано с тем, что все сигналы у вас неопределенны.

Давайте протестируем работу нашего отображения.

Следующий шаг: [Тестирование работы АРМ.](#)

9.4.3.6 Тестирование работы АРМ

Запустите модель и отображение. При этом на картинке у вас все картинки будут желтые (сигналы не определены).



Рис. 1 Неопределенные состояния насосов и индикаторов.

Попробуйте установить в WinDecont, в соответствии с нашей таблицей сигналов ([таблица сигналов](#)), единицы в дискретные состояния насосов. При этом отображение изменится.

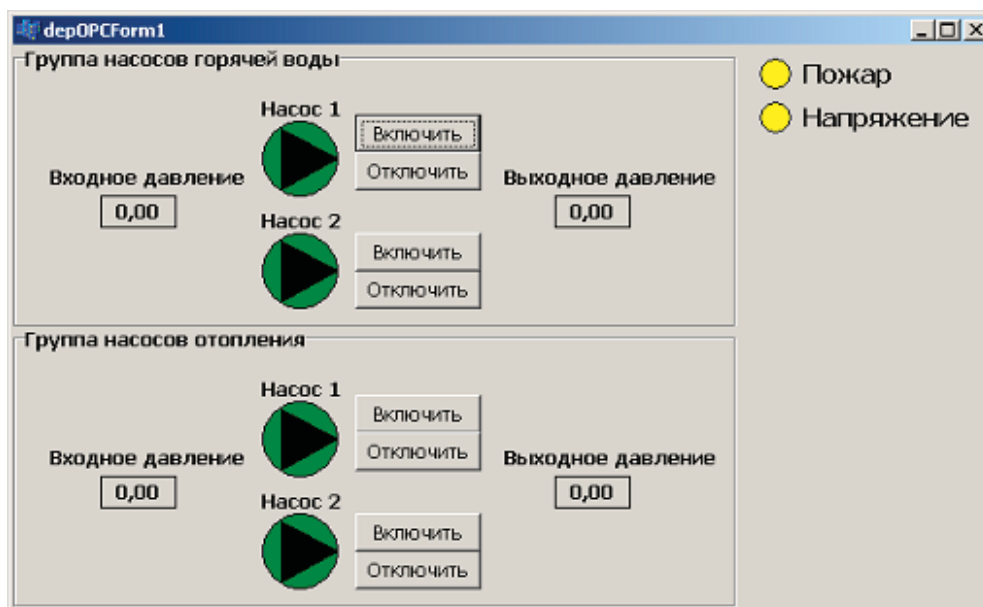


Рис. 2 Насосы включены.

Отключите, запишите 0, вторые насосы обеих групп.

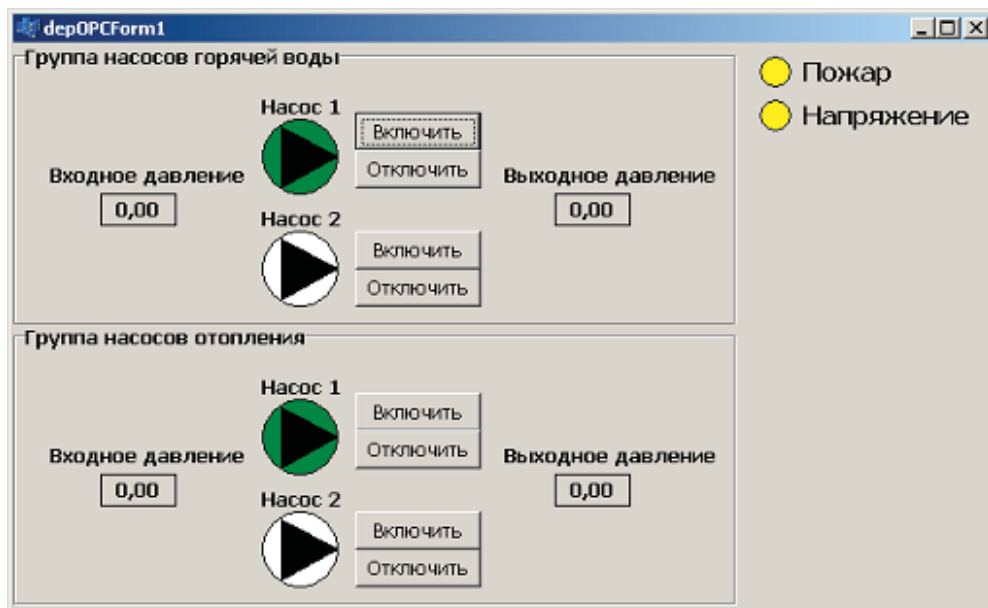


Рис. 3 Первые насосы включены, вторые отключены.

Замечательно! Состояние насосов отображается корректно. Давайте проверим работу индикаторов пожара и напряжения. Запишите 0 (нормальное состояние) в соответствующие дискреты.

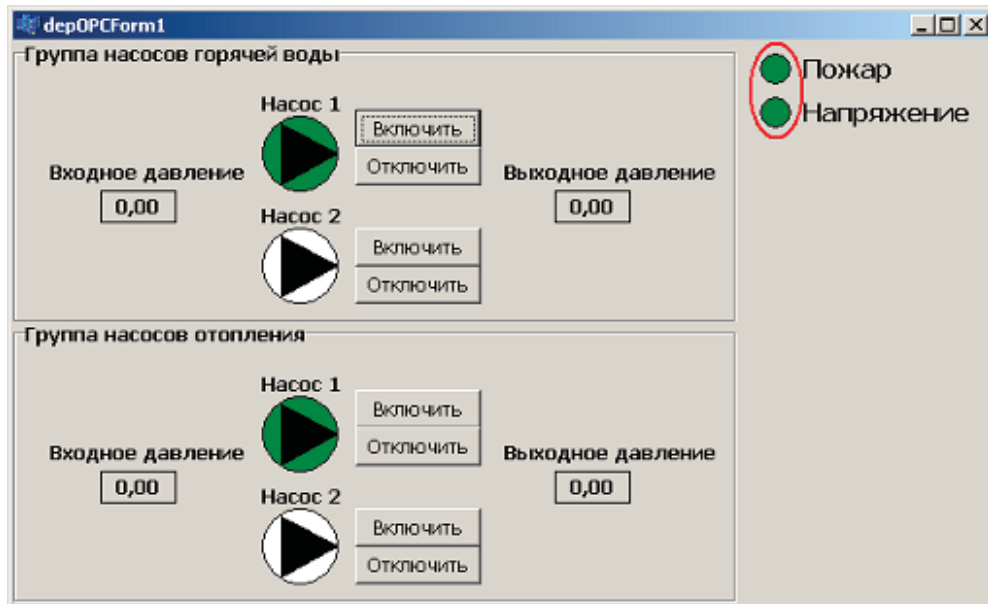


Рис. 4 Индикаторы пожара и напряжения в нормальном состоянии.

Индикаторы отработали как надо, теперь запишите 1 (аварийное состояние).

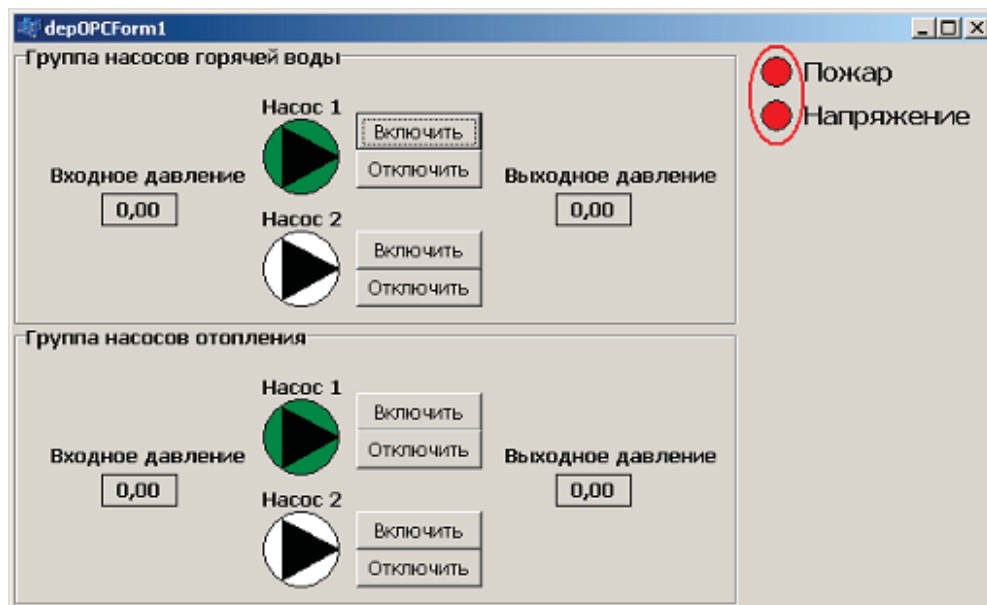


Рис. 5 Индикаторы показывают аварию.

Индикаторы работают как надо. Последнее, что мы должны проверить это отображение значений входных и выходных давлений. Запишите в соответствующие аналоги любые числа, например 123,11.

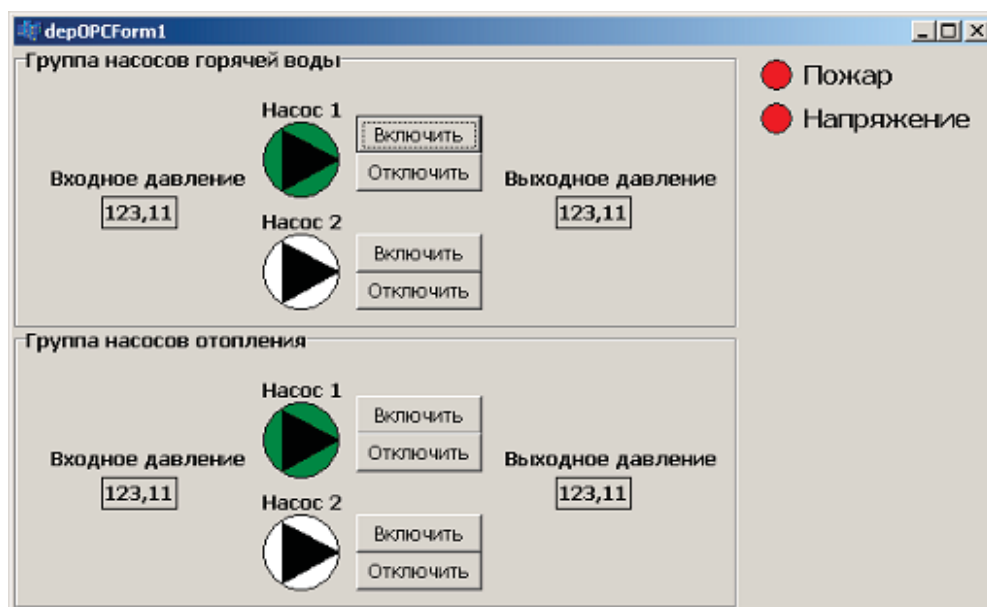


Рис. 6 Отображение значений входных и выходных давлений.

Поздравляю! наше отображение работает в соответствии с заданием на АРМ, и мы можем сдавать его заказчику.

Следующий шаг: [Итоги](#).

9.4.3.7 Итоги

Пора подвести итоги, вы:

1. Ознакомились и владеете всем необходимым инструментарием для создания АРМ.
2. Освоили процесс создания законченного проекта по автоматизации рабочего места диспетчера.
3. Создали модель, отображение, привязали отображение к модели и провели тестирование работы вашей системы.

Вы можете быть довольны: вы собственными силами создали законченную систему АРМ диспетчера и можете, не отходя от компьютера, наблюдать за происходящим на вашем объекте. Попробуйте сформулировать себе новую задачу и реализовать ее, используя это руководство в качестве подсказки. Некоторые, более сложные вопросы, касающиеся работы с "Конструктором OPC-модели" и "OPC-дизайнером", остались за рамками данного руководства. Вы сможете их изучить в процессе вашей практической работы с инструментарием, выполняя конкретные задачи автоматизации.

9.4.4 Построение АРМ диспетчера. Пример 2.

Рассмотрим примеры использования следующих элементов и функций.

1. Тревоги.
2. Работа с пользователями.
3. Работа с оперативным журналом.
4. Тактовая функция модели.
5. Архивирование.

Следующий шаг: [Постановка задачи](#).

9.4.4.1 Постановка задачи

Будем использовать модель и АРМ диспетчера уже рассмотренного ранее примера теплового пункта.

- 1) При срабатывании датчика пожара или напряжения соответствующий кружок на мнемосхеме начинает мигать красным цветом. При нажатии кнопки «Квитировать» мигание прекращается. При пропадании сигналов пожара или напряжения до квитирования мигание соответствующих кружков продолжается, но уже зеленым цветом.
- 2) Доступ к управлению оборудованием и квитированию событий осуществляется только для зарегистрированных пользователей.
- 3) Информация о срабатывании датчиков пожара и напряжения, регистрациях пользователей отображается в оперативном журнале.
- 4) При превышении выходного давления выше порогового значения равного 10 начинает мигать соответствующая надпись на технологическом экране диспетчера.
- 5) Выполняется архивирование аналоговых значений входных и выходных давлений для групп насосов при изменении на величину большую, чем 0,1 (т.е. с величиной закругления 0,1).

Примерный вид технологического экрана, который должен получиться изображен на рис. 1.

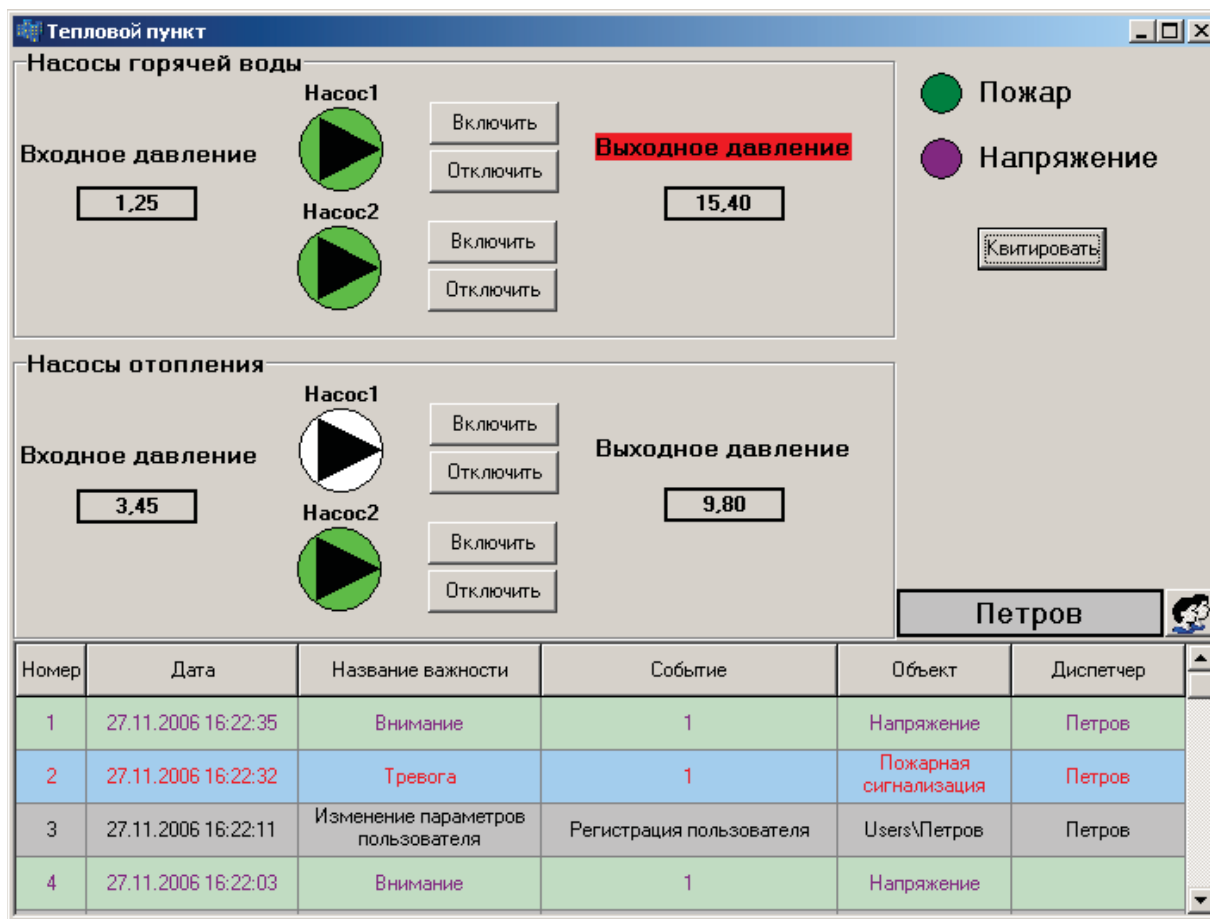


Рис.1. Общий вид технологического экрана.

Следующий шаг: [Создание модели](#).

9.4.4.2 Создание модели

9.4.4.2.1 Структура модели

Для реализации перечисленных выше функций дополнительно к уже имеющимся нам потребуются следующие элементы:

Элемент тревоги (SignalAlarm)

Используется для управления флагом «Внимание» элементов модели «Пожар (Fire)» и «Напряжение (Voltage)».

Элемент (Users)

Используется для управления пользователями.

Элемент (OperLog)

Используется для работы с оперативным журналом.

Элемент (AlarmOutPress)

Принимает значение 1 при превышении выходного давления выше порогового. Изменяется в тактовой функции модели, используется для мигания надписи «Выходное давление».

Следующий шаг: [Элемент тревоги](#).

9.4.4.2.2 Элемент тревоги

Добавим в главный тип СТР новый элемент (назовем его "SignalAlarm"). В качестве типа элемента выберем "alState" из библиотеки «Тревоги». Более подробно описание типов тревог можно посмотреть в документации по работе с моделью в разделе «Элементы и функции модели/Тревоги».

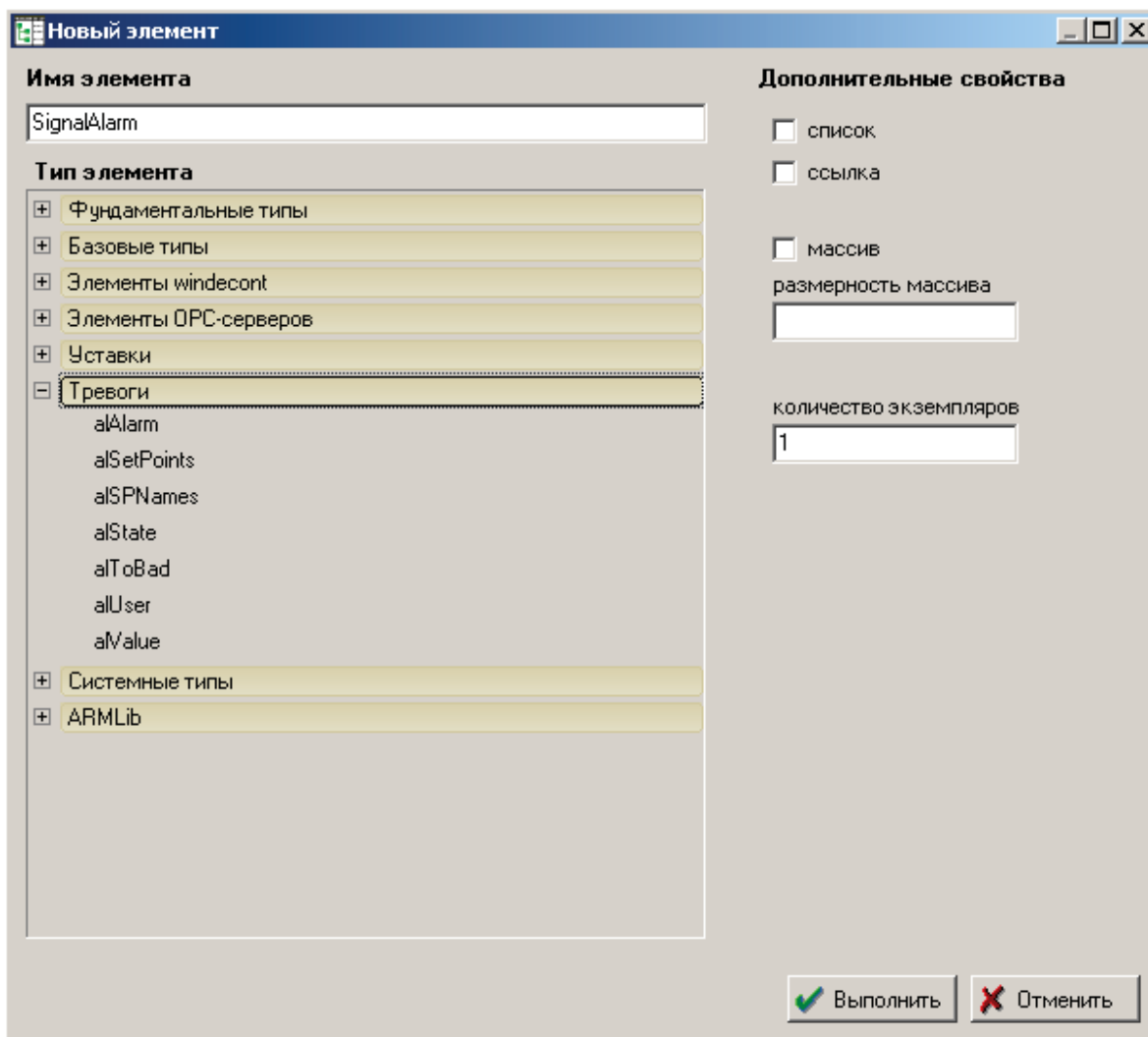


Рис. 1. Создание элемента тревоги (SignalAlarm).

В качестве одного из параметров только что созданного элемента "SignalAlarm" необходимо указать имя типа контролируемых элементов (TypeName). Такими элементами в нашем случае являются «Пожар (Fire)» и «Напряжение (Voltage)». Имя типа для этих элементов должно отличаться от имен типов всех остальных элементов, поэтому создадим новый тип "FVSignal", в качестве базового типа для него укажем "wdDIn".

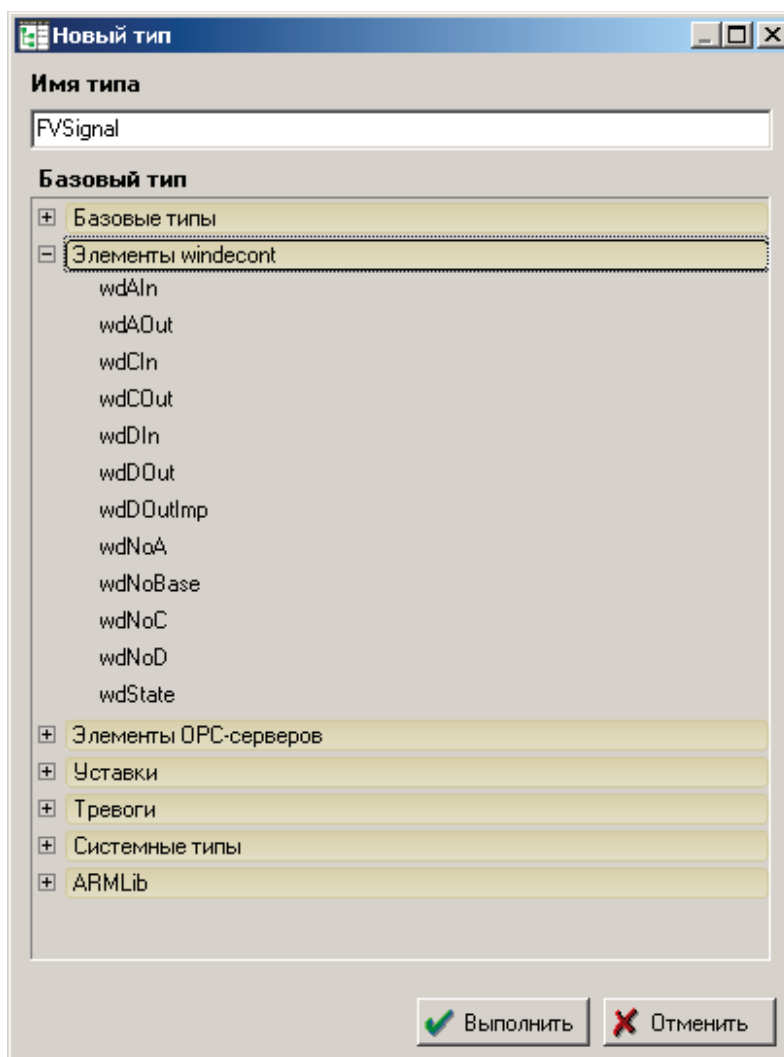


Рис. 2. Создание типа "FVSignal".

Сохраним этот тип, после этого для элементов модели "Fire" и "Voltage" изменим тип "wdDIn" на только что созданный тип "FVSignal" (в столбце «Тип» напротив названий "Fire" и "Voltage" щелкнем мышкой по значку с многоточием).

Описание элемента					
Имя	Тип	Значение	Название	Маска	Сохранять
[-] Тип СТР	от ilnt			<input type="checkbox"/>	<input type="checkbox"/>
[-] SignalAlarm	alState			<input type="checkbox"/>	<input type="checkbox"/>
[-] Fire	wdDIn	...	Пожарная сигнализация	<input type="checkbox"/>	<input type="checkbox"/>
[-] Voltage	wdDIn		Напряжение	<input type="checkbox"/>	<input type="checkbox"/>
[-] HotWaterGroup	PumpGroup		Группа насосов горя	<input type="checkbox"/>	<input type="checkbox"/>
[-] HeatingGroup	PumpGroup		Группа насосов отоп.	<input type="checkbox"/>	<input type="checkbox"/>

Рис. 3. Изменение типа элемента.

Откроется окно изменения типа элемента, в котором из библиотеки "ARMLib" выберем тип "FVSignal".

Далее для созданного элемента тревоги "SignalAlarm" необходимо указать все необходимые параметры. Значением параметра "Root" будет тип "СТР" (в этом узле будут найдены элементы типа "FVSignal"). Значением "TypeName" будет "FVSignal". Остальные параметры необходимо указать, как показано на рисунке:

Описание элемента					
Имя	Тип	Значение	Название	Маска	Сохранять
[-] Тип СТР	от ilnt			<input type="checkbox"/>	<input type="checkbox"/>
[-] SignalAlarm	alState			<input type="checkbox"/>	<input type="checkbox"/>
[-] List	tList<tBase>				
[-] Root	tLink<tBase>	(СТР)\		<input type="checkbox"/>	<input type="checkbox"/>
[-] TypeName	AnsiString	FVSignal			
[-] AutoReset	bool	True			
[-] AlarmControl	bool	True			
[-] CheckedState	int	1			
[-] Fire	FVSignal		Пожарная сигнализация	<input type="checkbox"/>	<input type="checkbox"/>
[-] Voltage	FVSignal		Напряжение	<input type="checkbox"/>	<input type="checkbox"/>
[-] HotWaterGroup	PumpGroup		Группа насосов горя	<input type="checkbox"/>	<input type="checkbox"/>
[-] HeatingGroup	PumpGroup		Группа насосов отоп.	<input type="checkbox"/>	<input type="checkbox"/>

Рис. 4. Установка параметров элемента тревоги "SignalAlarm".

"AutoReset" – (True) для автоматического сброса значения элемента тревоги в «0», если контролируемые элементы ("Fire", "Voltage") не в аварийном состоянии.

"AlarmControl" – (True) для установки признака «Внимание» у контролируемых элементов.

"CheckedState" – (1) определяет значение контролируемого элемента, при котором он находится в аварийном состоянии, т.е. у него установлен признак «Внимание».

Признак «Внимание» у элемента тревоги сбрасывается автоматически при сбросе этого признака у контролируемых элементов.

Следующий шаг: [Элемент для работы с пользователями](#)

9.4.4.2.3 Элемент для работы с пользователями

Добавим в главный тип СТР новый элемент (назовем его "Users"). В качестве типа элемента выберем "tUsers" из библиотеки «Системные типы».

Подробное описание всех параметров можно посмотреть в документации по работе с моделью в разделе «Элементы и функции модели/Пользователи».

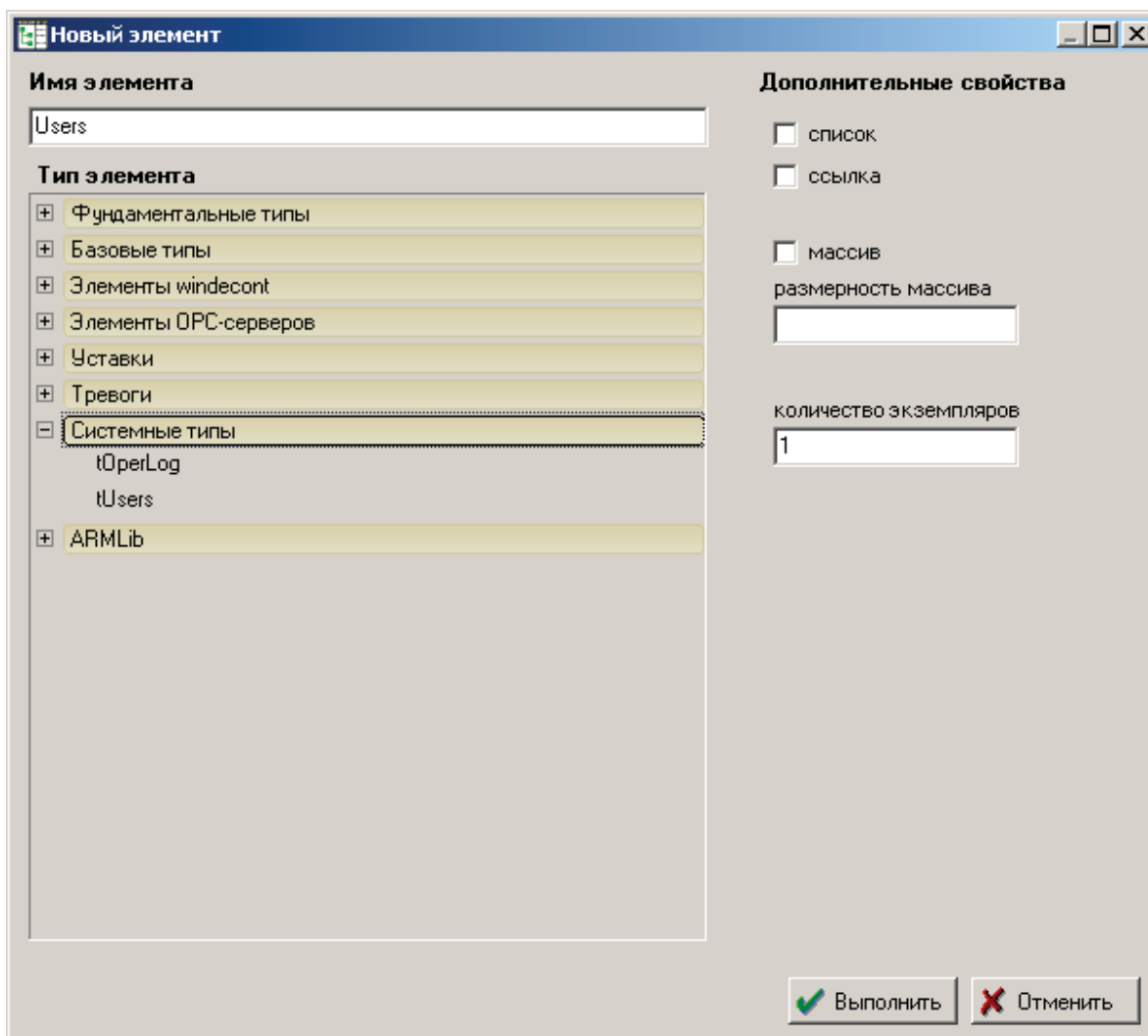


Рис. 1. Создание элемента "Users".

Для только что созданного типа "Users" укажем все необходимые параметры.

"Count" – (10) количество возможных пользователей системы.

"AdminPwd" – (useradmin) пароль администратора.

Для нашего примера выберем, например, три категории пользователей ("0", "1", "2") с названиями: «Просмотр», «Администратор», «Диспетчер». Пользователь с правами администратора сможет только добавлять/удалять других пользователей, но не сможет управлять оборудованием, т.е. включать/выключать насосы, квитировать аварийные события и т.п. В данном случае только пользователю с правами диспетчера разрешены функции управления системой.

Заполним для нашего примера все необходимые поля в типе "Users" как показано на рисунке.

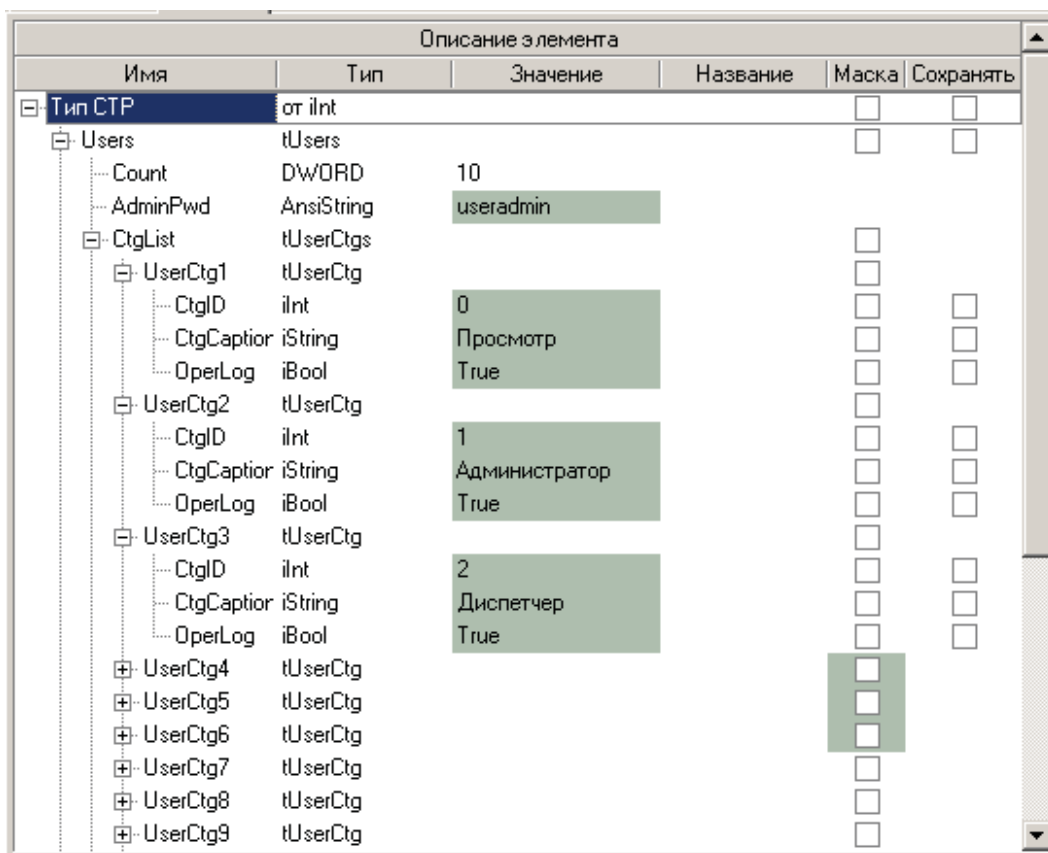


Рис. 2. Установка параметров типа "Users".

Следующий шаг: [Элемент для работы с оперативным журналом.](#)

9.4.4.2.4 Элемент для работы с оперативным журналом

Подробное описание элементов и функции по работе оперативного журнала находится в документации по работе с моделью в разделе «Элементы и функции модели/Оперативный журнал».

Добавим в главный тип новый элемент (назовем его "OperLog"). В качестве типа элемента выберем "tOperLog" из библиотеки «Системные типы».

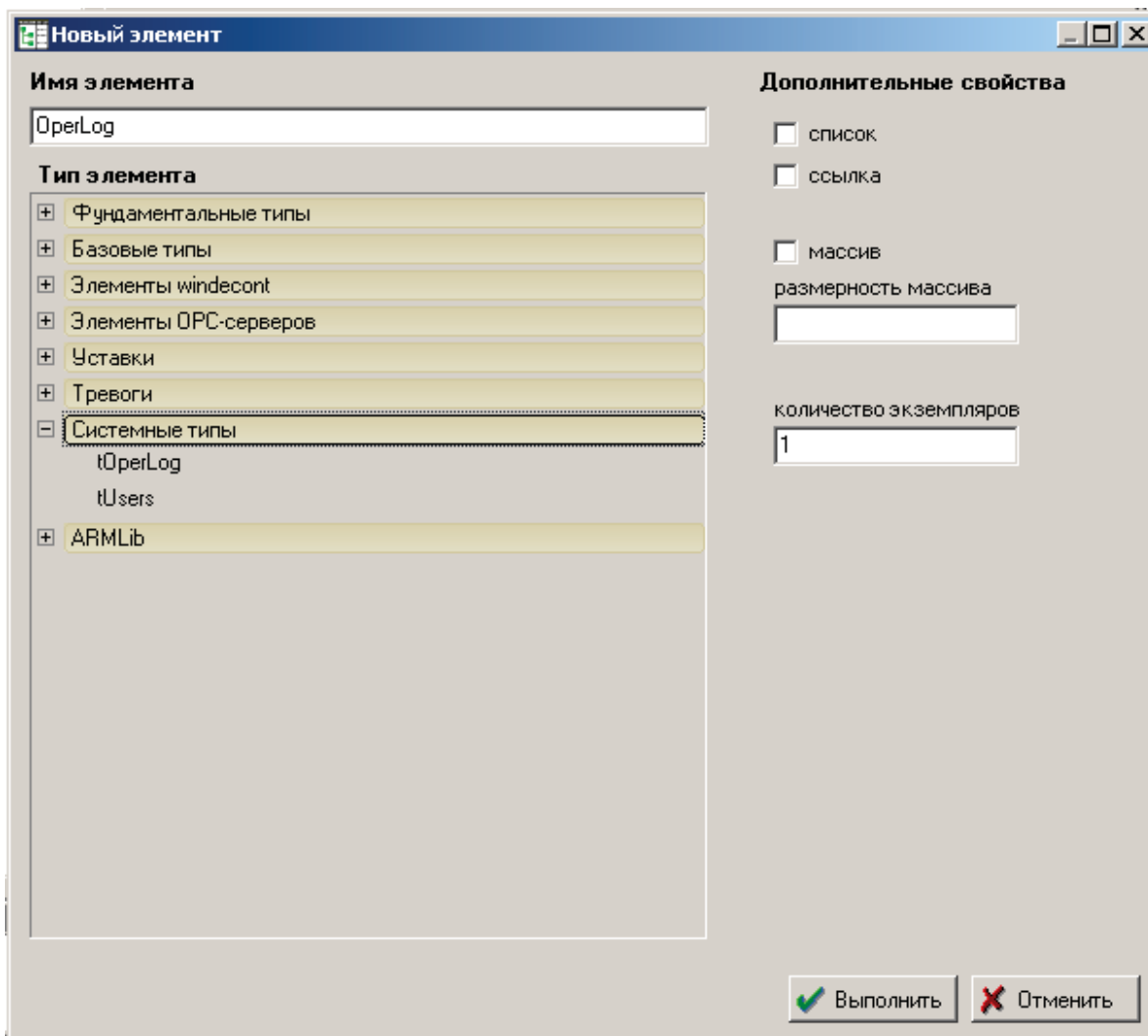


Рис. 1. Создание оперативного журнала.

Укажем, какие события должны отображаться в оперативном журнале кроме регистрации пользователей. В нашем примере это события «Пожар» и «Напряжение». Т.е. при поступлении сигнала «1» с датчиков пожара и напряжения кроме мигания соответствующих кружков на мнемосхеме должна появиться соответствующая запись в оперативном журнале.

Сначала нужно заполнить справочник событий оперативного журнала. Откроем справочник как показано на рисунке.

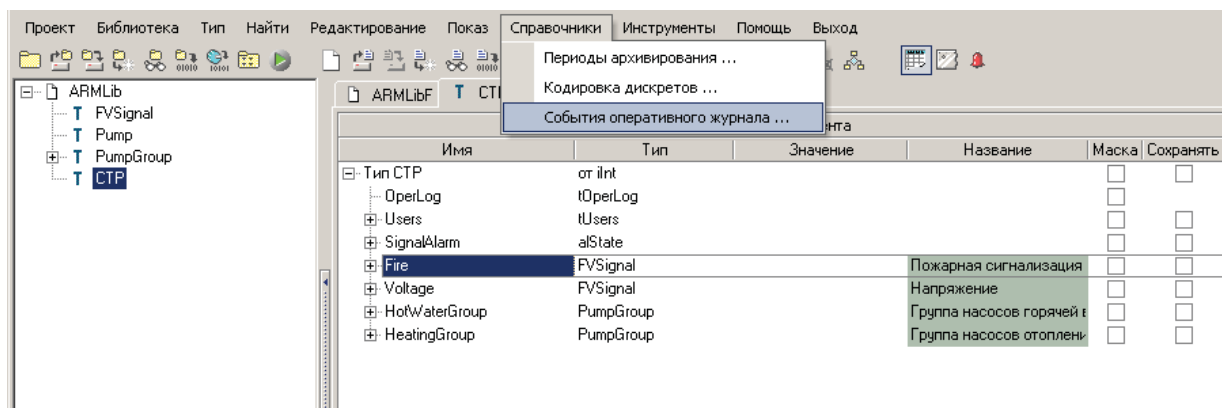


Рис. 2. Открытие справочника событий оперативного журнала.

Добавим записи «Внимание» и «Тревога» и укажем для них «Важность», например, 300 и 500.

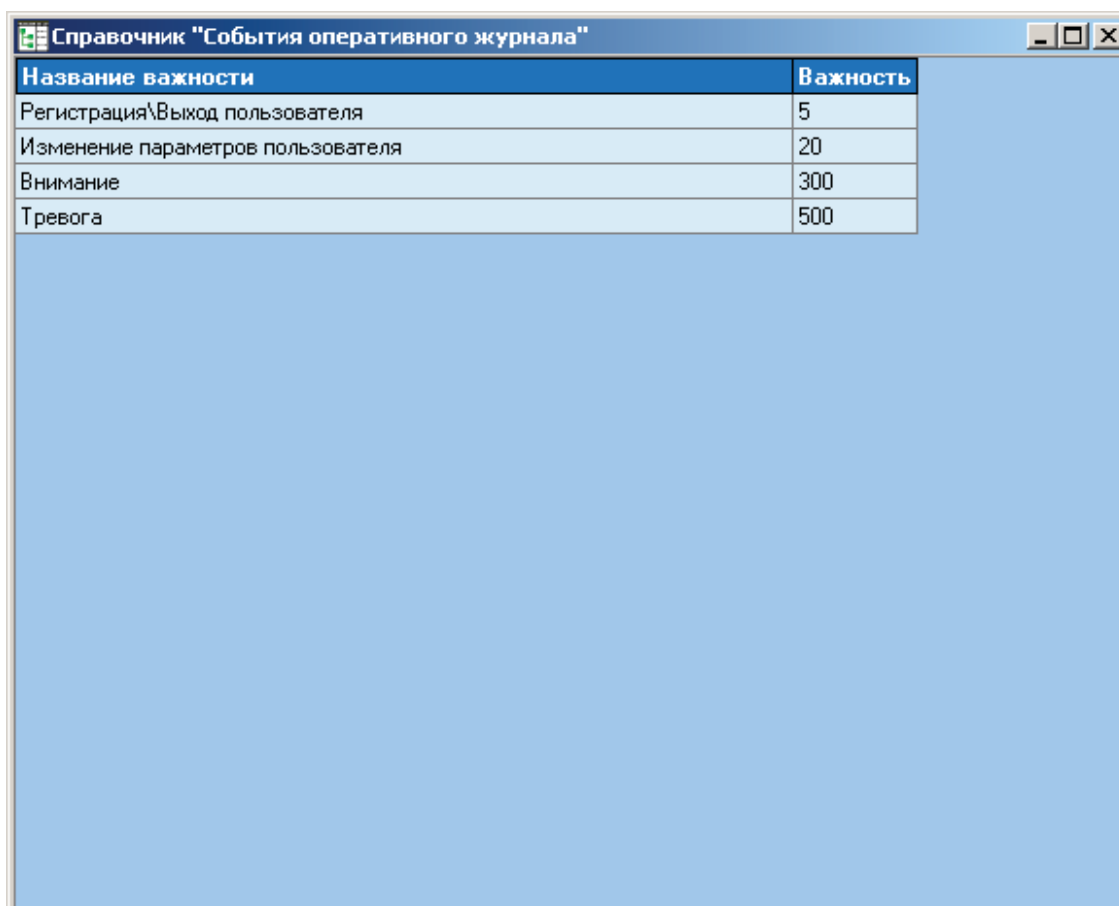



Рис. 3. Заполнение справочника оперативного журнала.

После этого закроем справочник и на панели инструментов нажмем на значок , для всех элементов модели должны

появятся параметры оперативного журнала. Для элементов "Fire" и "Voltage" укажем «Важность» 500 и 300 соответственно, названия важности появятся автоматически.

Описание элемента						Оперативный журнал		
Имя	Тип	Значение	Название	Маска	Сохранять	Важность	Название важно...	Квитировать
[-] Тип СТР	ot iInt			<input type="checkbox"/>	<input type="checkbox"/>	0		<input type="checkbox"/>
[-] OperLog	tOperLog			<input type="checkbox"/>				
[+] Users	tUsers			<input type="checkbox"/>	<input type="checkbox"/>	0		<input type="checkbox"/>
[+] SignalAlarm	alState			<input type="checkbox"/>	<input type="checkbox"/>	0		<input type="checkbox"/>
[+] Fire	FVSignal		Пожарная сиг	<input type="checkbox"/>	<input type="checkbox"/>	500	Тревога	<input type="checkbox"/>
[+] Voltage	FVSignal		Напряжение	<input type="checkbox"/>	<input type="checkbox"/>	300	Внимание	<input type="checkbox"/>
[+] HotWaterGr	PumpGroup		Группа насос	<input type="checkbox"/>	<input type="checkbox"/>	0		<input type="checkbox"/>
[+] HeatingGrou	PumpGroup		Группа насос	<input type="checkbox"/>	<input type="checkbox"/>	0		<input type="checkbox"/>

Рис. 4. Параметры оперативного журнала.

Для работы оперативного журнала необходимо в параметрах работы модели указать псевдоним хранилища. Как это сделать показано ниже в разделе [«Архивирование»](#).

Следующий шаг: [Тактовая функция модели](#).

9.4.4.2.5 Тактовая функция модели

Для реализации задачи мигания надписей «Выходное давление» для обеих групп насосов добавим в тип "PumpGroup" новый элемент (назовем его "AlarmOutPress"). В качестве типа элемента выберем "iInt" из библиотеки базовых типов.

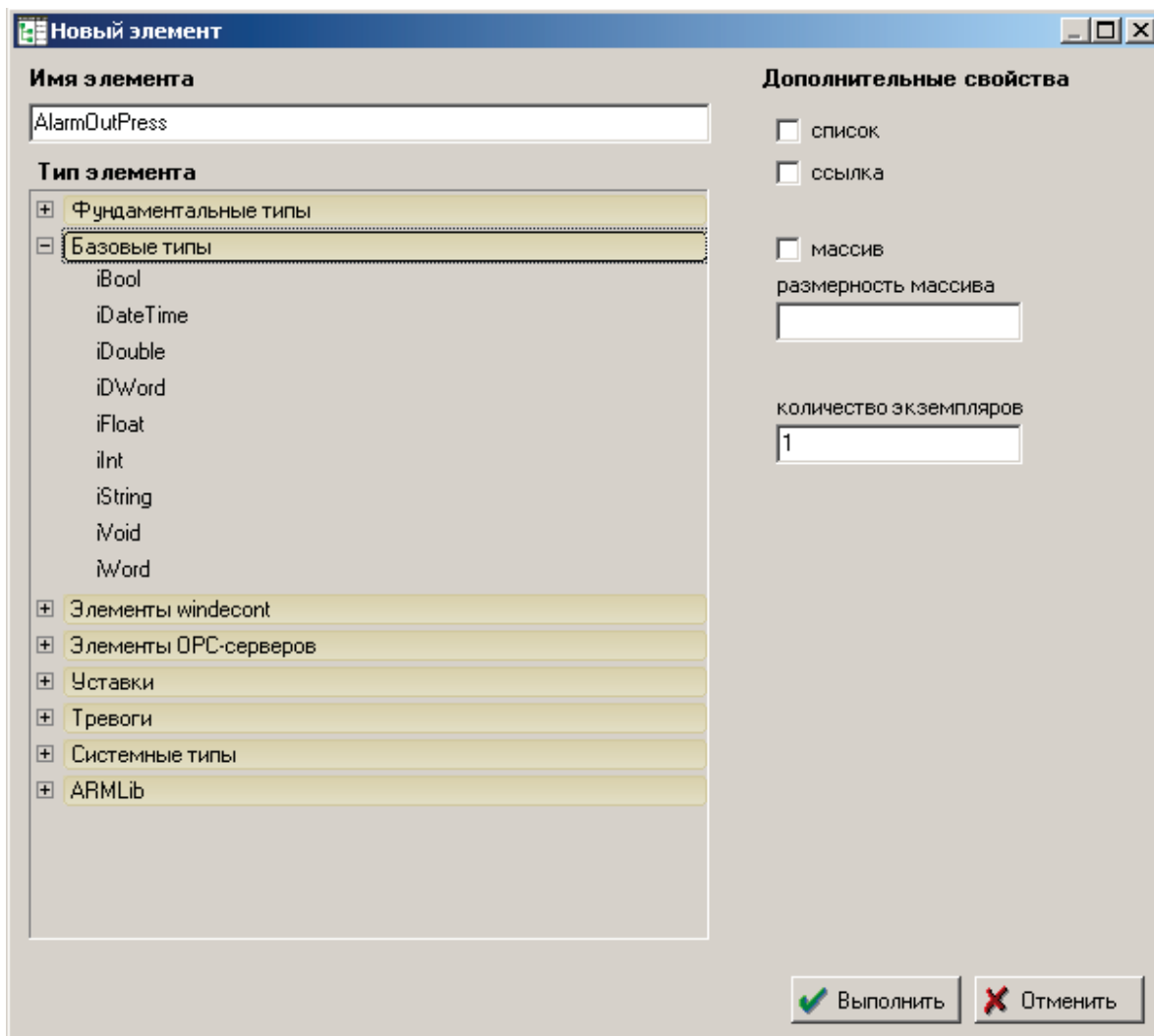



Рис. 1. Создание нового элемента.

На каждом такте работы модели (в тактовой функции) будет проверяться условие превышения выходного давления выше порогового значения (в нашем случае 10) и элементу модели "AlarmOutPress" будет присваиваться значение «1», в противном случае «0».

Для добавления тактовой функции в тип "PumpGroup" выберем мышкой этот тип на панели набора библиотек и типов модели, на панели инструментов щелкнем мышкой по значку «Новая функция» () . После этого появится окно добавления функции, в котором выберем параметры как показано на рисунке.

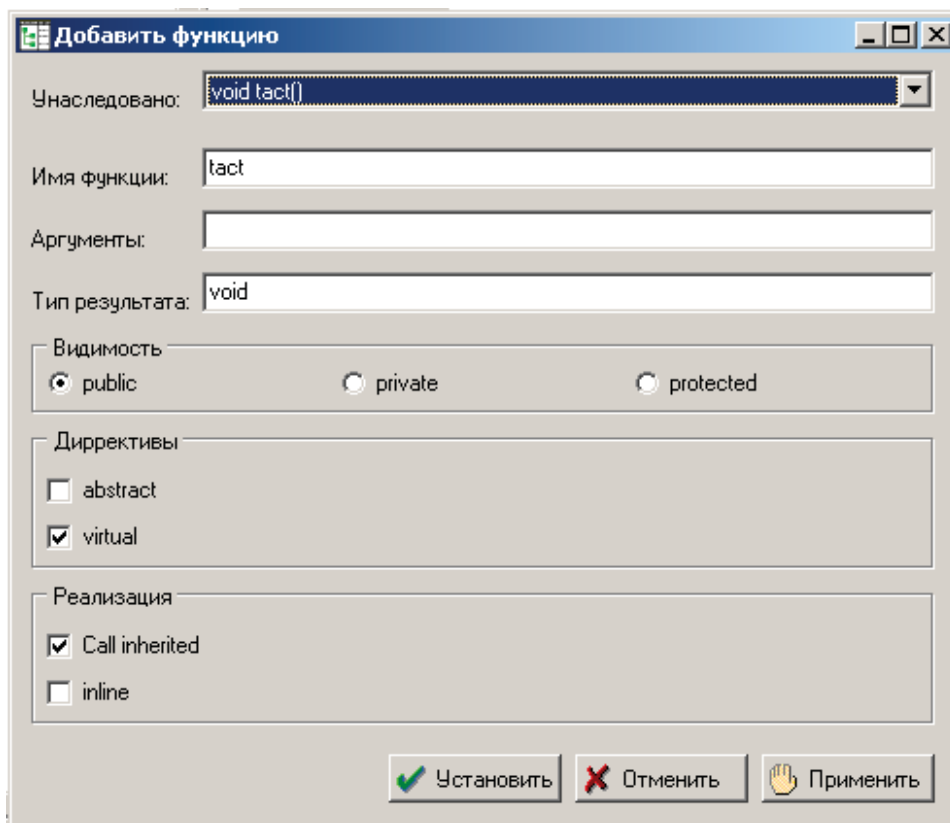


Рис. 2. Добавление функции.

После добавления имя функции появится в типе "PumpGroup". Перейдем в окно редактирования функции (два раза щелкнем мышкой по имени "tact"). Реализуем условие проверки значения выходного давления и присвоения «1» элементу модели "AlarmOutPress", например, как показано на рисунке.

Более подробно описание используемых функций можно посмотреть в документации по работе с моделью в разделе «Программирование».

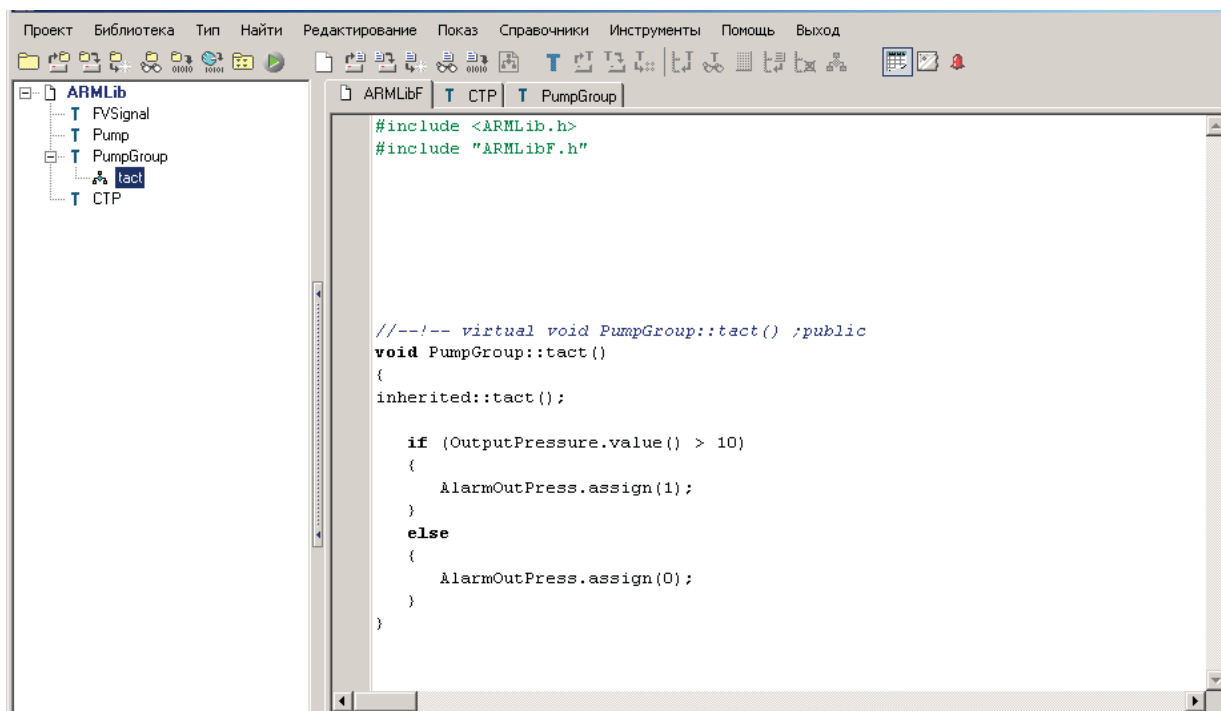


Рис. 3. Пример тактовой функции.

Следующий шаг: [Архивирование](#).

9.4.4.2.6 Архивирование

Сначала необходимо в программе «Менеджер хранилища» создать хранилище с псевдонимом, например, «Архив_СТР». В установках нашей модели укажем это хранилище.

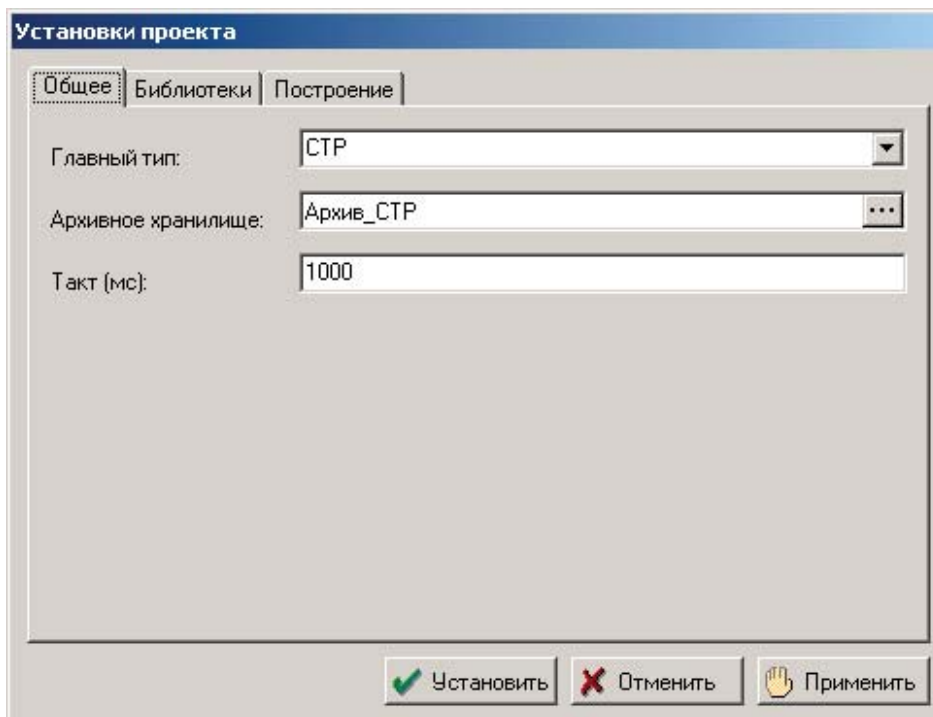



Рис. 1. Установки модели.

В конструкторе модели на панели инструментов щелкнем по значку «Параметры архивирования» (). Для элементов "InputPressure" и "OutputPressure" типа "PumpGroup" в столбце «А» поставим галочку, а в столбце «Заглубление» поставим 0,1 (т.е. архивировать аналоги при изменении на величину большую 0,1).

Описание элемента		Архивирование											
Имя	Название	Д	А	С	АС	АМ	АМн	АМх	СМ	СР	З...	Т...	Т...
[-] Тип PumpGroup		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0	
[-] Mode	Режим работы	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0	
[-] InputPressure	Входное давление	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0.1	
[-] OutputPressure	Выходное давление	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0.1	
[-] Pump_1	Насос1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0	
[-] Pump_2	Насос2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0	
[-] AlarmOutPres:		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	0	

Рис.2. Архивирование аналогов по изменению.

В программе "WinDecont" в установках модели выберем имя нашего архива и поставим галочку в столбце «Архивировать».

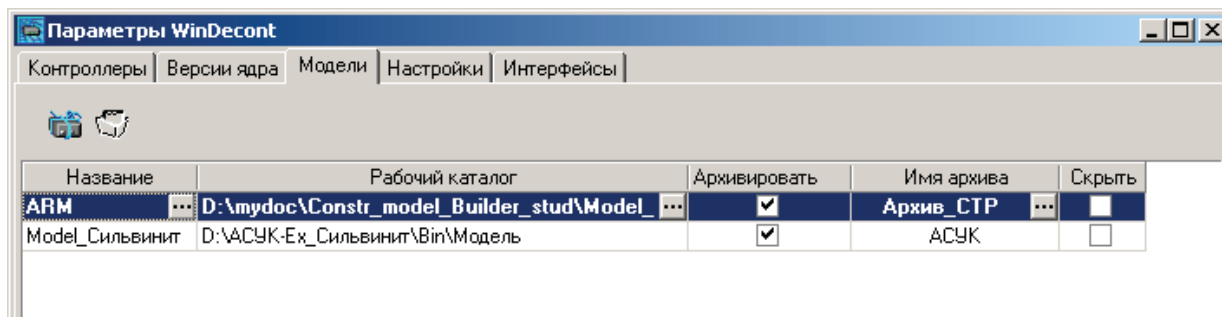


Рис. 3. Настройка модели в программе "WinDecont".

Более подробно информацию по архивированию можно посмотреть в документации по работе с моделью в разделе «Элементы и функции модели\Архивирование».

Следующий шаг: [Создание отображения.](#)

9.4.4.3 Создание отображения

9.4.4.3.1 Индикаторы пожара и напряжения

Из ранее рассмотренного примера мы уже хорошо знаем, как осуществляется привязка свойств компонентов изображения к элементам модели, поэтому мы не будем останавливаться на этом подробно.

К уже существующим привязкам изображений индикаторов пожара и напряжения добавим еще по одной привязке. В окне выбора свойств для динамизации выберем вкладку «Специальные свойства» и на ней свойство «Мигание».

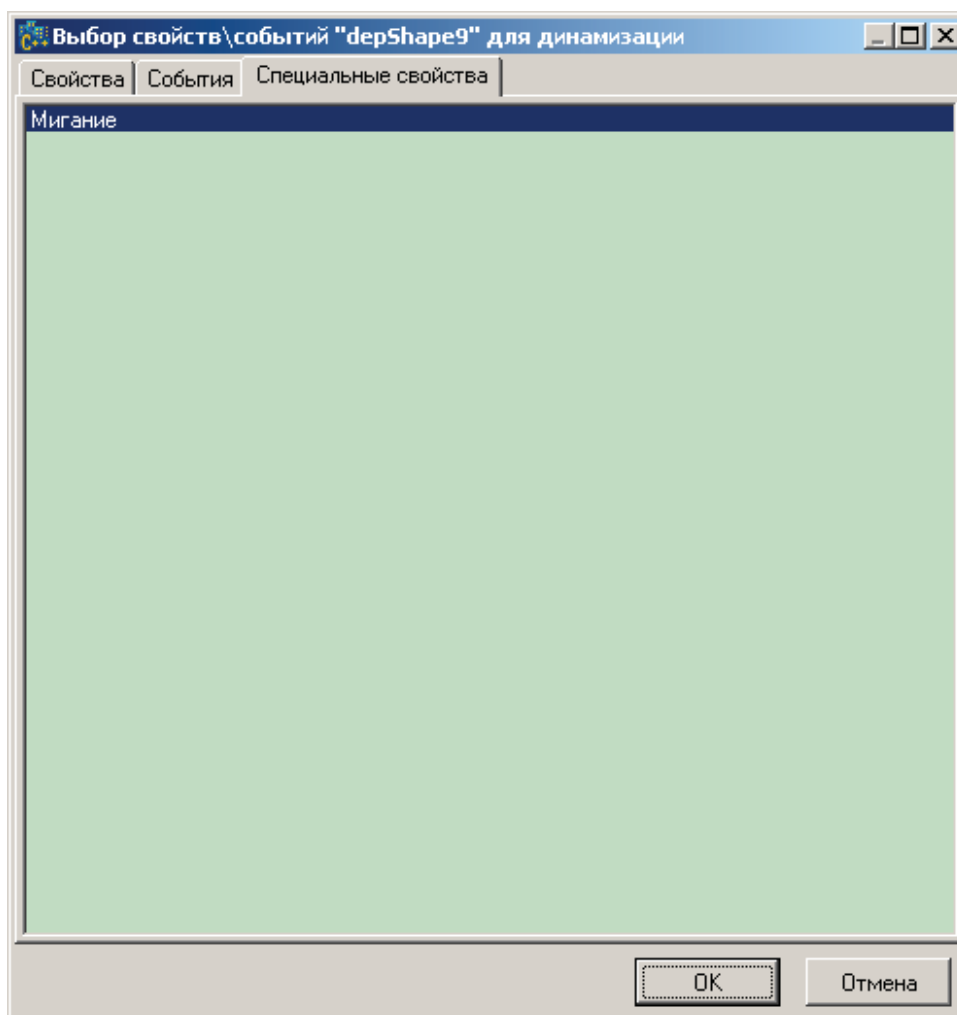


Рис. 1. Выбор свойства "Мигание".

К этой привязке добавим свойство "Brush.Color". В окончательном виде обе привязки к элементу модели "Fire" должны иметь вид, как показано на рисунках.

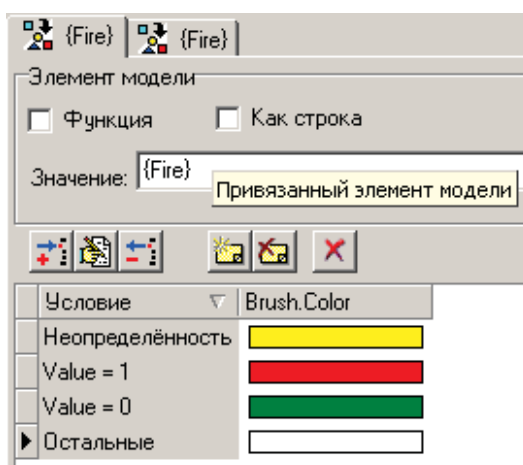


Рис. 2. Привязка к элементу модели "Fire".

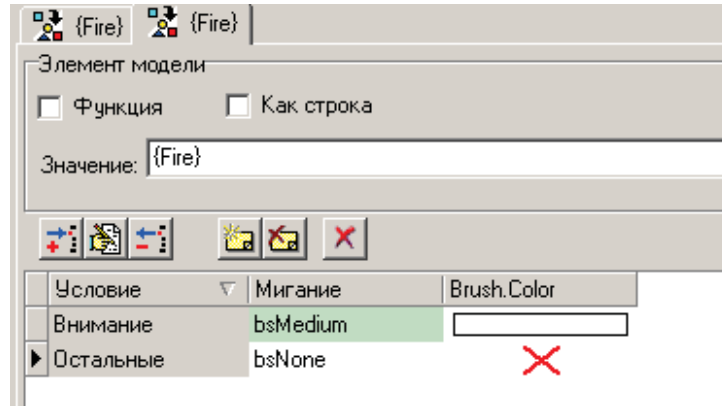


Рис. 3. Привязка свойства "Мигание" к элементу модели "Fire".

Такая комбинация привязок обеспечит мигание на мнемосхеме красным цветом на белом фоне индикатора пожара в случае поступления сигнала «1» с датчика пожара.

Как мы видим, мигание индикатора осуществляется при взведенном флаге «Внимание», который характеризует тревожное состояние элемента модели. Установление этого флага в элемент "Fire" мы обеспечили добавлением в модель элемента тревоги "SignalAlarm" и установлением параметра "CheckedState" в «1». Что мы не сделали, так это автоматический сброс флага «Внимание» у элемента модели "Fire", т. е. индикатор пожара будет продолжать мигать зеленым цветом при пропадании сигнала «1» с датчика пожара. Но нам это и не нужно, т.к. диспетчер должен увидеть не только наличие сигнала пожара, но и факт его возникновения в прошлом и вручную квитировать эти события путем нажатия на кнопку «Квитировать» на мнемосхеме.

Добавим эту кнопку на мнемосхему и для нее в редакторе OPC состояний по событию «Левое нажатие клавиши мыши» установим запись значения «6» в элемент модели "SignalAlarm\Control", что требуется для сброса бита «Внимание» у элемента тревоги и у всех контролируемых элементов.

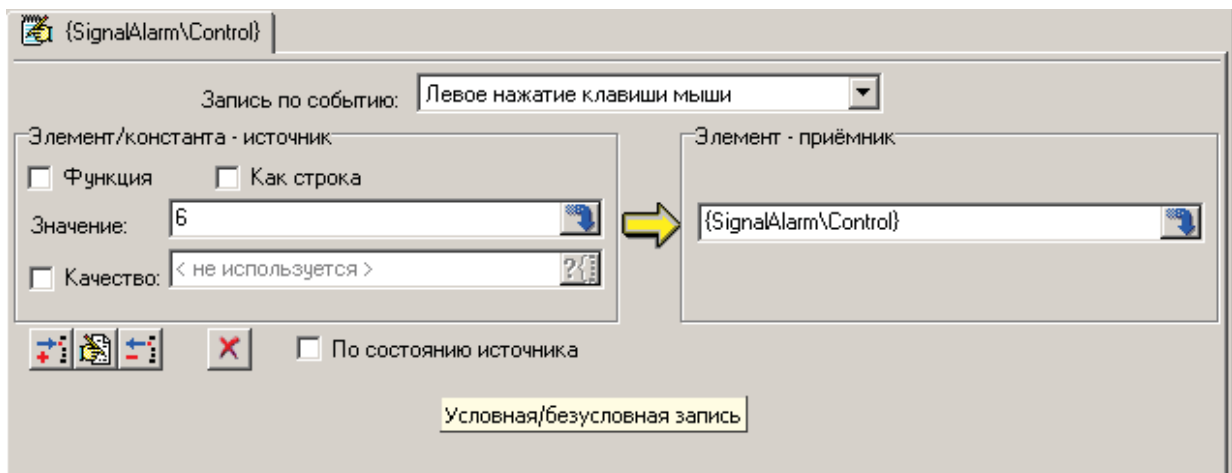


Рис. 4. Сброс тревоги.

После нажатия на кнопку «Квитировать» индикаторы пожара и напряжения мигать перестанут.

Следующий шаг: [Компонент для работы с пользователями](#).

9.4.4.3.2 Компонент для работы с пользователями

Подробное описание компонента для работы с пользователями можно найти в документации по работе с моделью в разделе «OPC компоненты для C++ Builder\Дополнительные компоненты\Работа с пользователями».

Из закладки "DEP OPC" поместим на форму компонент "TdepAdmin". Свойства этого компонента укажем как на рисунке.

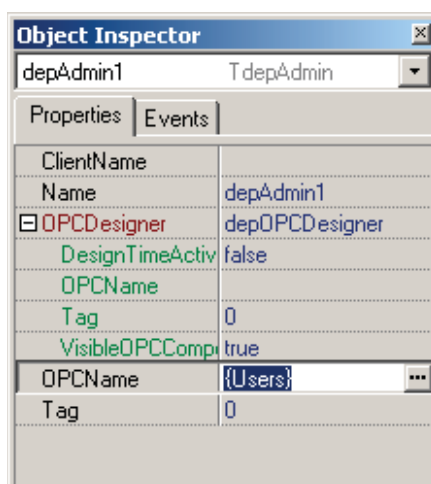


Рис. 1. Свойства компонента TdepAdmin.

В качестве "OPCName" нужно выбрать элемент модели, отвечающей за работу с пользователями, в нашем случае это элемент "Users".

Далее добавим на мнемосхему кнопку, при нажатии на которую, появляется диалог регистрации пользователей. Возьмем, например, компонент "TSpeedButton" из закладки "Additional". Для красоты в свойстве "Glyph" укажем какой-нибудь рисунок как в нашем примере. В инспекторе объектов на закладке "Events" напротив события "OnClick" (нажатия на кнопку) два раза щелкнем мышкой, появится редактор обработки этого события.

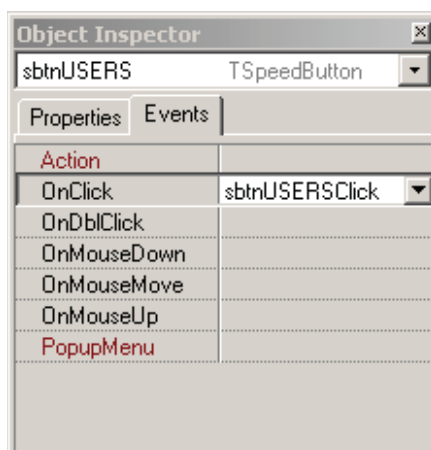
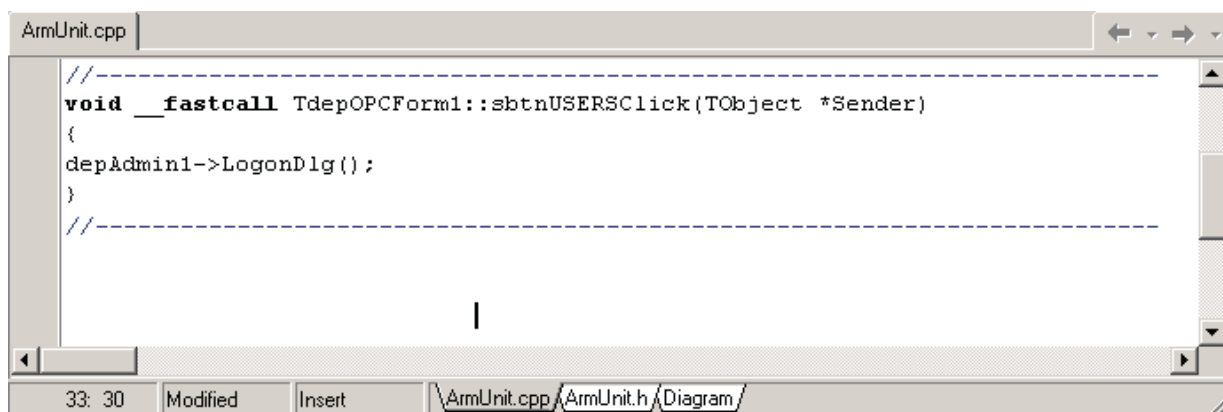


Рис. 2. Вызов редактора обработки события OnClick.



```
ArmUnit.cpp  
//-----  
void __fastcall TdepOPCForm1::sbtnUSERSClick(TObject *Sender)  
{  
    depAdmin1->LogonDlg();  
}  
//-----  
|  
33: 30 Modified Insert \\ArmUnit.cpp\\ArmUnit.h\\Diagram/
```

Рис. 3. Обработка события OnClick.

В редакторе, как показано на рисунке, вызовем метод "LogonDlg()" нашего компонента "TdepAdmin" для работы с пользователями (имя этого компонента в нашем примере "depAdmin1"). Теперь при нажатии данной кнопки на мнемосхеме будет появляться диалог регистрации пользователей и только после регистрации пользователя с правами «Диспетчер» будет дана возможность управлять оборудованием, т. е. писать какие-либо значения в элементы модели.

Кроме того, хотелось бы видеть на мнемосхеме имя зарегистрированного в данный момент пользователя или, если никто не зарегистрирован, какую либо надпись, например «Просмотр». Для этого добавим компонент "TdepLabel" из закладки "DEP" на форму (имя компонента возьмем "depLabel_Users"). Далее вызовем обработчик события "OnLogon" компонента "TdepAdmin" и изменим, например, как показано на рисунке свойство "Caption" у "depLabelUsers".

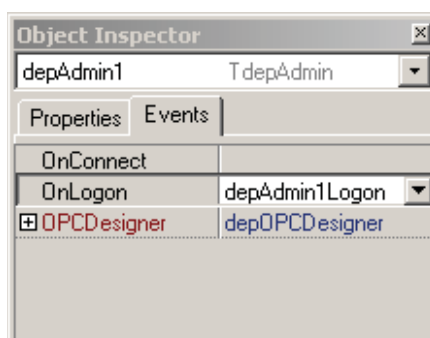
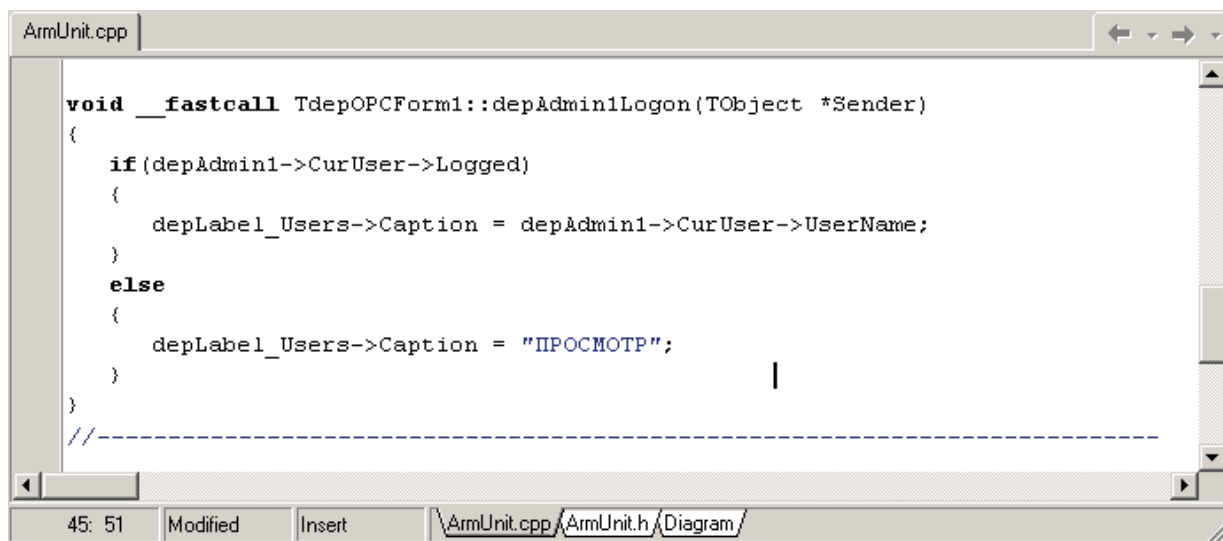


Рис. 4. Вызов редактора обработки события OnLogon.



```
void __fastcall TdepOPCForm1::depAdminLogon(TObject *Sender)
{
    if (depAdmin1->CurUser->Logged)
    {
        depLabel_Users->Caption = depAdmin1->CurUser->UserName;
    }
    else
    {
        depLabel_Users->Caption = "ПРОСМОТР";
    }
}
//-----
```

Рис. 5. Обработка события OnLogon.

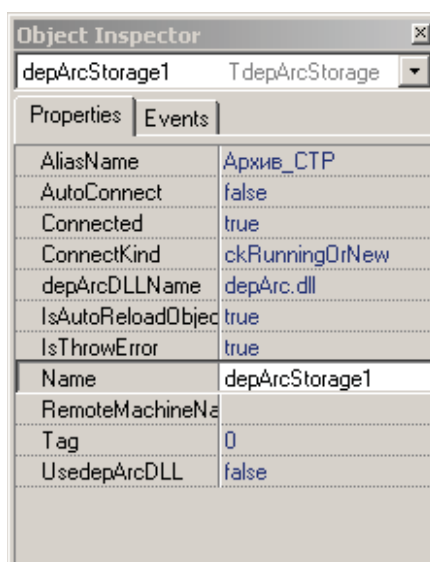
После этого на мнемосхеме будет отображаться имя текущего пользователя.

Следующий шаг: [Компонент для работы оперативного журнала.](#)

9.4.4.3.3 Компонент для работы оперативного журнала

Подробное описание компонента для работы с оперативным журналом можно найти в документации по работе с моделью в разделе «OPC компоненты для C++ Builder\Дополнительные компоненты\Работа с оперативным журналом».

Из закладки "DEP_ARC" поместим на форму компонент "TdepArcStorage". Свойства этого компонента укажем как на рисунке.



Object Inspector	
depArcStorage1 TdepArcStorage	
Properties	Events
AliasName	Архив_СТП
AutoConnect	false
Connected	true
ConnectKind	ckRunningOrNew
depArcDLLName	depArc.dll
IsAutoReloadObject	true
IsThrowError	true
Name	depArcStorage1
RemoteMachineName	
Tag	0
UsedepArcDLL	false

Рис. 1. Свойства компонента depArcStorage.

Обратим внимание на свойство "AliasName". Значением этого свойства должен быть псевдоним ранее созданного хранилища.

Далее из той же закладки добавим компонент "TdepOperLog".

В свойстве "OPCName" необходимо указать имя нашего элемента модели для работы с оперативным журналом "OperLog". Значением свойства "OLStorage" должно быть имя компонента "TdepArcStorage".

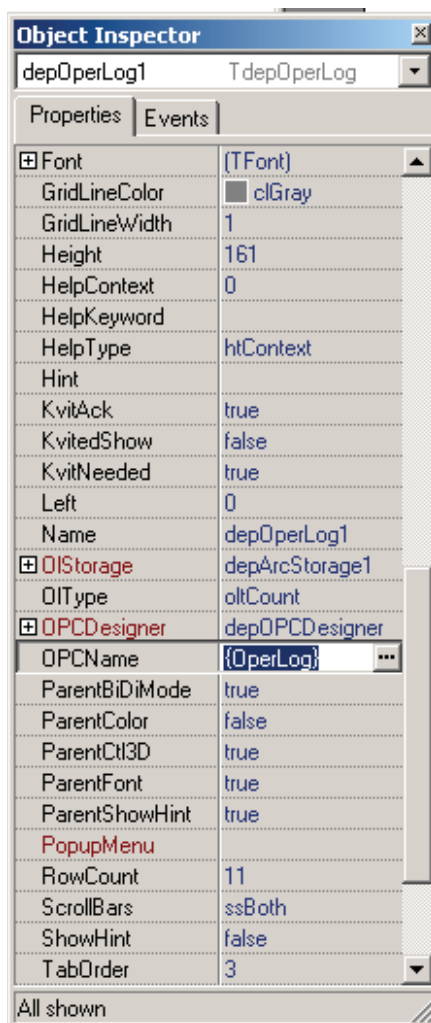


Рис. 2. Свойства компонента для работы с оперативным журналом.

Имеется ряд дополнительных свойств, с помощью которых, можно менять названия столбцов журнала, управлять списком столбцов и т.п.

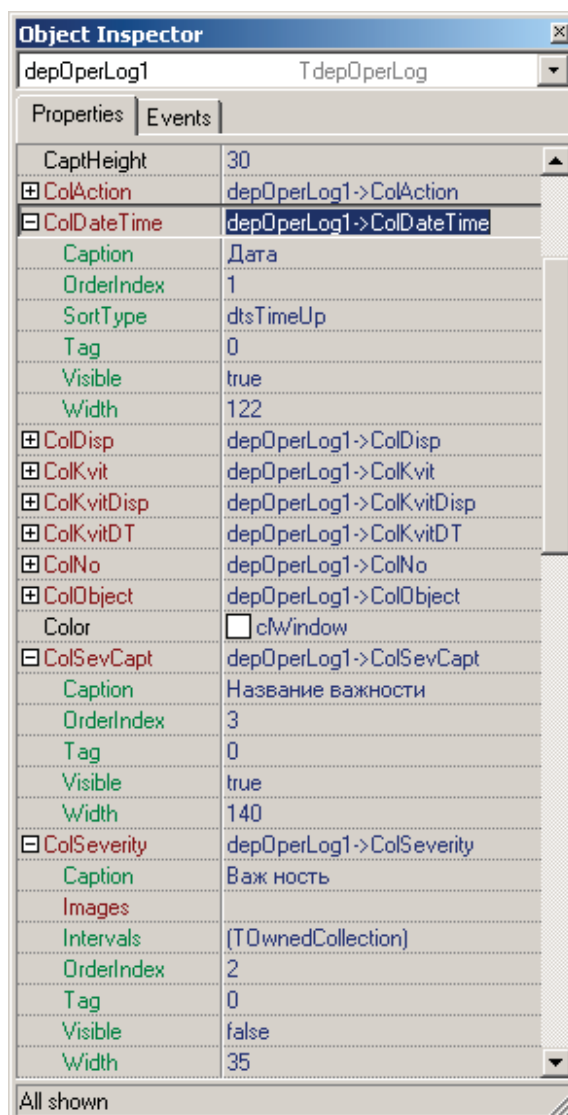


Рис. 3. Свойства компонента TdepOperLog.

Поменяем, например, цвет фона и цвет шрифта для различных типов записей. Если Вы еще не забыли, в нашем примере есть несколько видов записей, которые различаются свойством «Важность». Для них установим цвет фона и шрифта через свойство "ColSeverity\Intervals" как показано на рисунке.

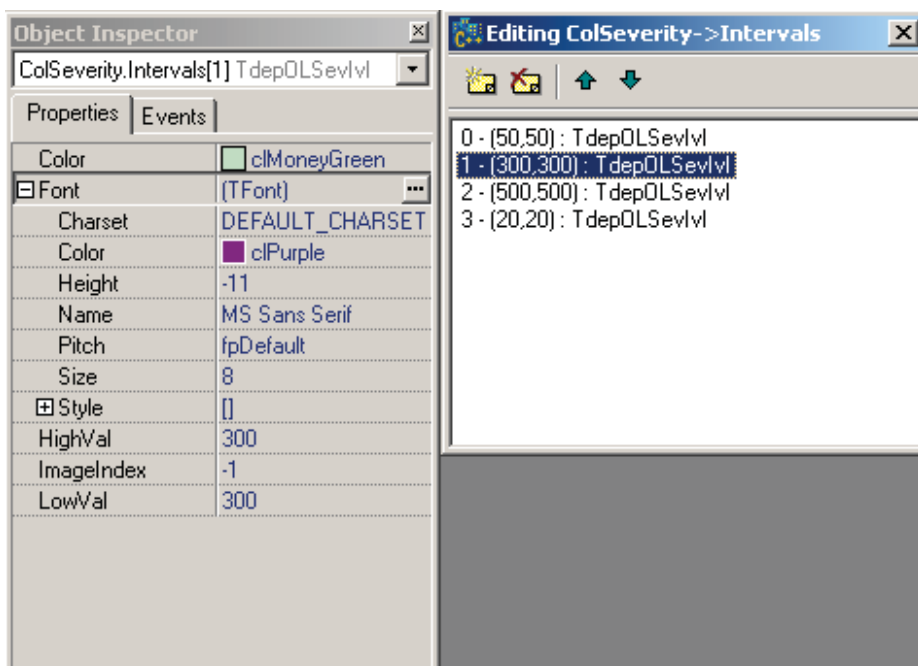


Рис. 4. Изменение стиля отображения записей журнала.

Наш проект готов и можем приступить к его тестированию.

Следующий шаг: [Тестирование работы APM.](#)

9.4.4.3.4 Дополнения

9.4.4.3.4.1 Квитирование Оперативного Журнала

В оперативном журнале предусмотрена возможность квитирования события. Квитирование происходит при двойном щелчке по строке события в оперативном журнале, если значение свойства `KvitNeeded` равно **true**. Однако изначально квитирование события в оперативном журнале никак не связано с квитированием этого события в модели. Для установки этой связи необходимо произвести ряд дополнительных действий.

Программный доступ к записям оперативного журнала осуществляется с помощью элемента `SrtList` оперативного журнала – это список структур типа `TdepArcLog`. Он определен в файле `depArc_TLB.hpp` следующим образом:

```
struct TdepArcLog
{
public:
    System::TDateTime DateTime;
    int Kind;
    WideString Action;
    WideString Object_;
    WideString Disp;
    WideString Comment;
    short Category;
    WideString OPCName;
```

```
Word Kvit;  
WideString KvitDisp;  
System::TDateTime KvitDateTime;  
int Id;  
int Stub;  
};
```

Каждая такая структура соответствует событию в оперативном журнале. Нас интересует поле OPCName – OPC-имя элемента, вызвавшего событие. Подробнее о составе структуры TdepArcLog можно прочитать в документации по работе с моделью в разделе «OPC компоненты для C++ Builder\Дополнительные компоненты\Работа с оперативным журналом».

1. Квитирование кнопкой «Квитировать».

При таком квитировании необходимо программно квитировать соответствующие события в оперативном журнале. Для этого предусмотрена функция TdepOperLog::Kvit(int ARecIndex), определенная в файле depOperLog.cpp. Схема действий следующая: проводится поиск по всему оперативному журналу событий с нужными OPC-именами, затем для таких событий вызывается функция Kvit. В приведенном выше примере достаточно фильтровать события по категории.

2. Квитирование из оперативного журнала.

Добавим обработчик двойного щелчка по оперативному журналу со следующим кодом:

```
KvitFromOperLog(depOperLog1->Row-1);
```

Свойство Row возвращает номер выделенной строки.

Функцию KvitFromOperLog(int index) определим следующим образом:

```
void TfrmMain::KvitFromOperLog(int index)  
{  
    Variant Var;  
    TdepOPCQuality Quality;  
    AnsiString s =  
((TdepArcLog*)(depOperLog1->SrtList->Items[index]))->OPCName;  
    //чтение старого значения  
    depOPCDesigner->ReadFromItem(AAlias,s,Var,Quality);  
    //взвод бита внимания  
    Quality.SetBitAlarmed(false);  
    depOPCDesigner->WriteToItem(AAlias,s,Var,Quality);  
    //функция квитирования одинаковых элементов  
    KvitClones(((TdepArcLog*)(depOperLog1->SrtList->Items[index]))  
->Object_);  
}
```

Функция KvitClones() необходима для квитирования повторяющихся в оперативном журнале элементов и может иметь следующий вид:

```
void TfrmMain::KvitClones(AnsiString Obj)  
{  
    AnsiString s;  
    for(int i=0;i<depOperLog1->RowCount-1;i++)  
    {
```

```

s=((TdepArcLog*)(depOperLog1->SrtList->Items[i]))->Object_;
if(s==Obj) depOperLog1->Kvit(i+1);
}
}

```

9.4.4.4 Тестирование работы АРМ

После первого запуска модели и отображения мнемосхема будет выглядеть следующим образом.

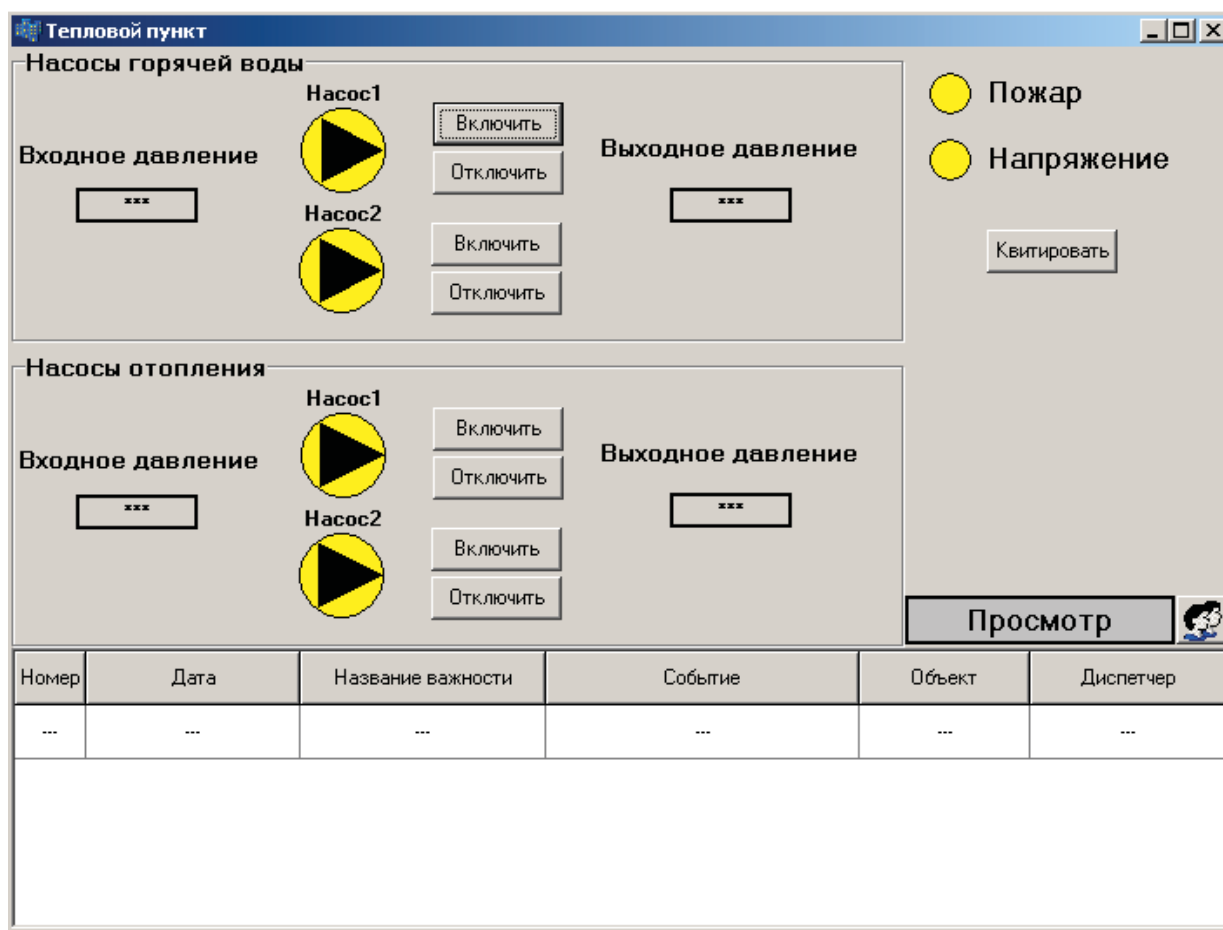


Рис. 1. Неопределенное состояние всех сигналов.

Установим какие-нибудь значения дискретных и аналоговых сигналов в соответствии с таблицей сигналов.

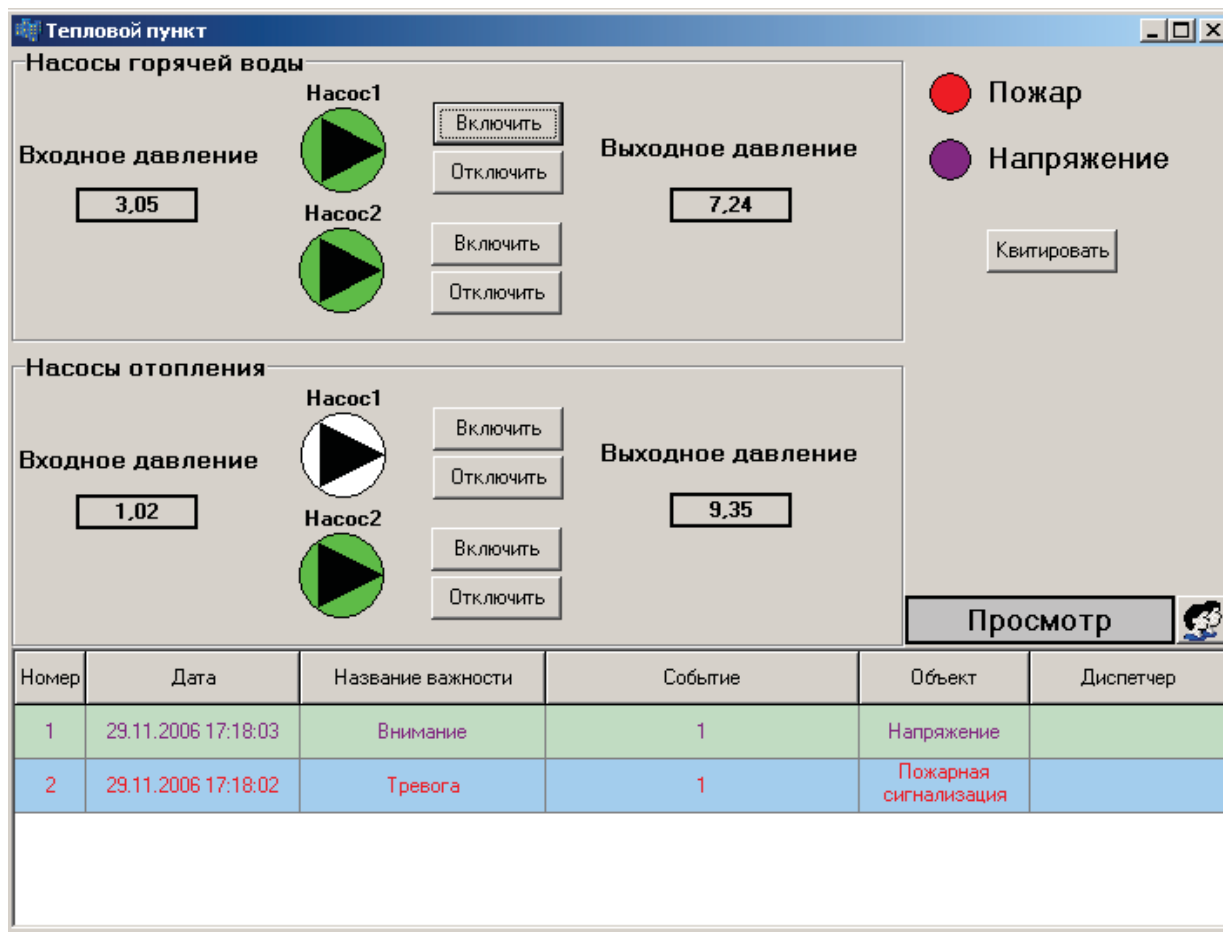


Рис. 2. Все сигналы определены.

Появились сигналы о состоянии насосов, с датчиков давления и аварийные сигналы пожара и напряжения. Также, появились записи в оперативном журнале.

На рисунке не видно мигание индикаторов пожара и напряжения, но в реальной ситуации они должны мигать с чередованием цветов красного/бардового и белого. Сейчас нажатие на кнопку «Квитировать» не приведет к квитированию мигания и не работают кнопки включения/выключения насосов, т.к. нет зарегистрированного пользователя с правами диспетчера.

Нажмем на кнопку работа с пользователями, зайдём под именем «Администратор» и добавим новых пользователей, одному из которых дадим права диспетчера. (если Вы еще не забыли, то пароль администратора "useradmin").

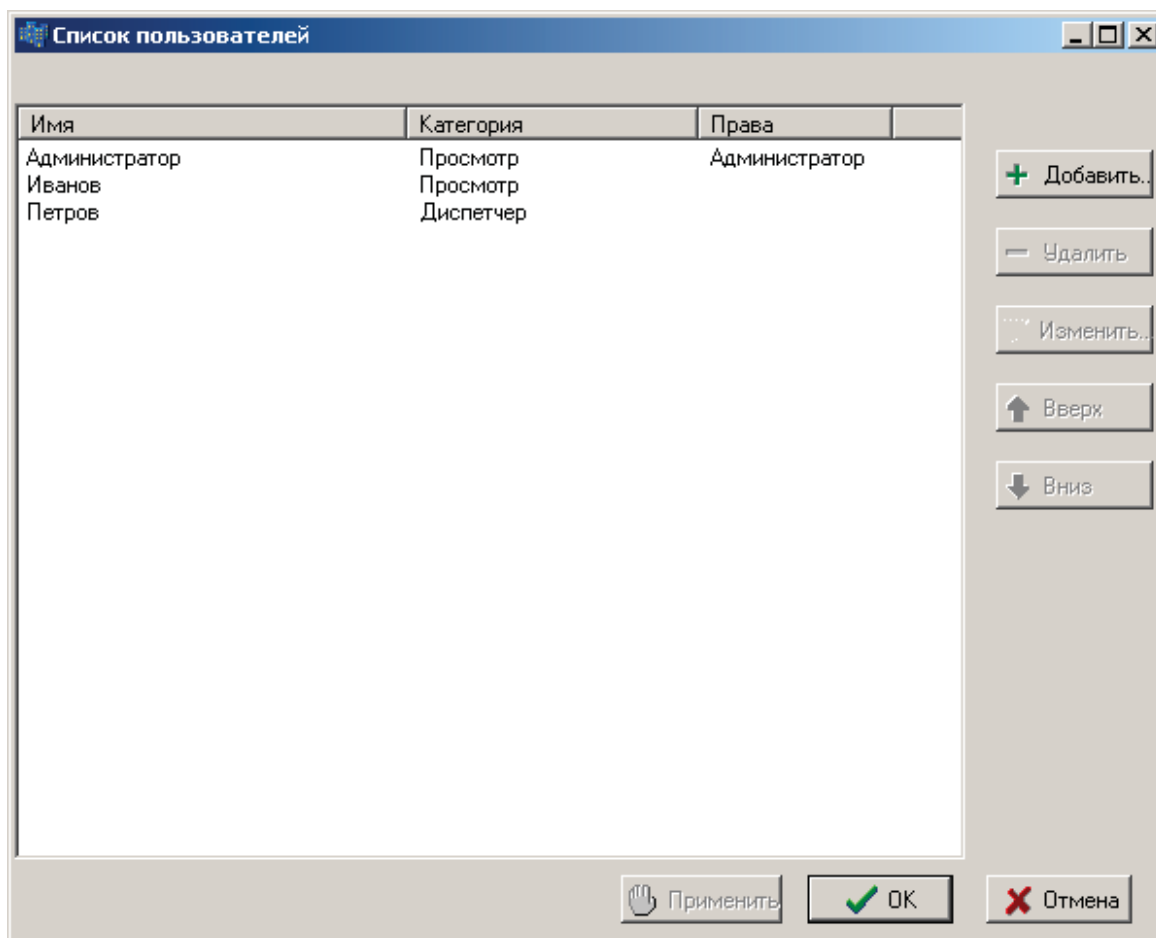


Рис. 3. Окно добавления пользователей.

После выхода пользователя Администратор и регистрации пользователя с правами диспетчера нажатие на кнопку «Квитировать» приводит к квитированию мигания индикаторов пожара и напряжения. Также, нажатие на кнопки включения/выключения насосов приводит к записи в базу параметров WinDecont соответствующих значений. Вся информация по добавлению/удалению и регистрациях пользователей добавилась в оперативный журнал.

Можно проверить мигание надписи «Выходное давление» при превышении давления выше критического (в нашем случае 10).

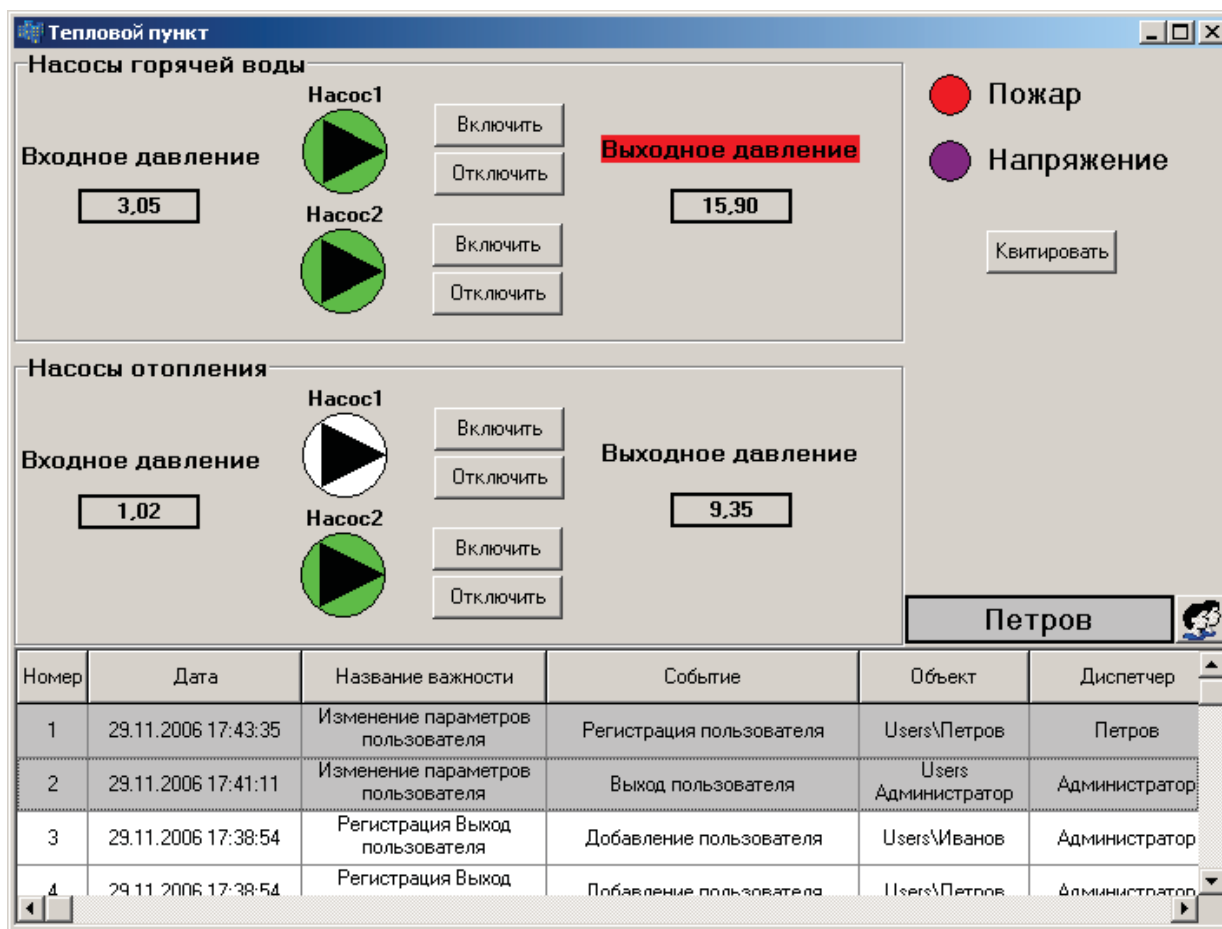


Рис. 4. Отображение действий с пользователями и подсветка надписи "Выходное давление".

Через программу «Просмотр архивов» можно посмотреть архивирование аналогов.

Теперь мнемосхема работает в соответствии с требованиями.

9.5 OPC компоненты для C++Builder

9.5.1 Введение

Разработанный набор компонентов предназначен для создания автоматизируемых рабочих мест (АРМ) контроля и управления оборудованием, используя спецификацию OPC Data Access (обмен текущими данными).

OPC (OLE for Process Control) – набор повсеместно принятых спецификаций предоставляющих универсальный механизм обмена данными в системах контроля и управления.

OPC технология обеспечивает независимость потребителей от наличия или отсутствия драйверов или протоколов, что позволяет выбирать оборудование и программное обеспечение, наиболее полно отвечающее реальным потребностям бизнеса.

В спецификации OPC для обмена данными определены два компонента - OPC-клиент и OPC-сервер.

OPC-сервер - программа, получающая данные во внутреннем формате устройства или системы и преобразующая эти данные в формат OPC. OPC-сервер является источником данных для OPC-клиентов.

OPC-клиент - программа, принимающая от OPC-серверов данные в формате OPC и преобразующая их во внутренний формат устройства или системы.

OPC-клиент общается с OPC-сервером посредством строго определенных в спецификации интерфейсов, что позволяет любому OPC-клиенту общаться с любым OPC-сервером. Однажды созданный OPC-сервер позволяет подключать устройство к широкому кругу программного обеспечения (SCADA системам, HMI и др.) поддерживающего спецификацию OPC.

В данном программном продукте реализован OPC-клиент, работающий с любым OPC-сервером(поддерживаются спецификации OPC-серверов до 3.0 включительно).

Разработка основы приложения ведётся в визуальной среде разработки CodeGear RAD Studio. Для разработки обычных мнемосхем от пользователя не требуется глубоких знаний языка программирования Delphi или C++ и среды программирования. Требуются лишь базовые знания интерфейса CodeGear RAD Studio. Вся работа по созданию проекта ведётся во встроенном визуальном редакторе. Редактор позволяет динамизировать опубликованные свойства компонентов: задать зависимость значения свойства от значения элемента OPC сервера. Под словом "динамизация" здесь и далее будет подразумеваться правила, по которым какой-либо объект изменяет свое состояние или влияет на поведение других объектов, в зависимости от состояния тэгов(элементов) OPC-сервера. Для динамизации могут быть использованы любые компоненты среды CodeGear RAD Studio, а также сторонних производителей и разработанные самим пользователем. Для любого компонента не требуется написания какого либо дополнительного кода. Возможна динамизация SVG-графики (Scalable Vector Graphics) для любого SVG-элемента.

Работа созданного приложения основана на взаимодействии с различными OPC серверами.

Работа OPC компонентов тестировалась с OPC серверами:

- **Dep.Model.x.** Разработка модели OPC-сервера (OPC-модели) ведётся в [Конструкторе OPC модели](#). Для более удобной разработки проекта отображения рекомендуется сначала создать OPC-модель.
- **OPC.Dep.x.** Представляет собой OPC сервер базы параметров контроллера WinDecont. (см. описание WinDecont).

9.5.2 Что нового в версии 3 ?

В depComponentsPack 3 входит:

1. Development OPC ARM Expert (эксперт среды программирования CodeGear RAD Studio 2007&2009 для разработки автоматизированных рабочих мест)
2. Display SVG (отображение SVG графики. Компонент TdepSvgPanel).
3. Work with archive data (Работа с архивными данными. Компоненты TdepArcStorage, TdepArcPeriodicAnalogViewRead, TdepArcPeriodicCounterViewRead, TdepArcNoPeriodicAnalogViewRead, TdepArcNoPeriodicCounterViewRead, TdepArcEventViewRead, TdepArcLogBookOGViewRead, TdepArcViewX). Для их использования требуется установить дистрибутив "Работа с архивами" 6-ой версии.
4. Custom Components (Специализированные компоненты: TdepWavePlayer, TdepShape, TdepDateTimePanel).

Основные новые возможности (возможно не все):

- Поддержка SVG (сокращение от англ. Scalable Vector Graphics, масштабируемая векторная графика). Подробнее на <http://ru.wikipedia.org/wiki/SVG>.
- Для создания SVG файлов рекомендуется использовать открытый графический редактор Inkscape (<http://ru.wikipedia.org/wiki/Inkscape>). Компонент TdepSvgPanel работает с SVG-клонами как с отдельными объектами. OPC привязки для клона наследуются из оригинала, при динамизации в клоне можно только поменять OPC имя SVG-объекта и добавить новые привязки. Тем самым клоны можно рассматривать как подобие VCL-фреймов.
- Возможно создание OPC APM приложений как в среде C++Builder, так и в Delphi.

- Повысилась быстрота работы и объем памяти, выделяемый разрабатываемым приложением, за счет изменения внутренней структуры объектов.
- Вся динамизация хранится в файлах xml формата.
- Группы динамизации, для более быстрого задания привязок к типовым элементам.
- Вызов редактора динамизации из приложения (возможность исправить OPC динамизацию без среды программирования).
- Возможно быстрое подключение/отключение OPC динамизации (в среде программирования меню depOPC Design->OPC Config).
- Для любого динамизированного компонента возможно задать OPC имя (подложку). Раньше для этого использовался в версиях 2.x использовался компонент TdepOPCPanel.
- Для выбора OPC имени привязки используется более усовершенствованный OPC проводник.
- Вызов редактора OPC динамизации по "горячим клавишам".

Обратите внимание:

- Для данной версии требуется CodeGear RAD Studio 2007 или 2009. Разработку проекта можно вести в среде Delphi или C++Builder.
- Совместимость с предыдущими версиями (2.x) не поддерживается и вряд ли будет поддерживаться.
- depComponentsPack 3.0 распространяется по принципу "как есть". При этом не предусматривается никаких гарантий, явных или подразумеваемых. Вы используете его на свой собственный риск. Автор не отвечает за потери данных, повреждения, потери прибыли или любые другие виды потерь, связанные с использованием (правильным или неправильным) этой программы.

9.5.3 OPC проект отображения мнемосхемы

"OPC проектом" в контексте данной справки будем называть приложение, работающее с OPC серверами и созданное для автоматизации рабочего места.

9.5.3.1 Общие принципы создания мнемосхемы

Построение мнемосхемы состоит из трёх основных частей:

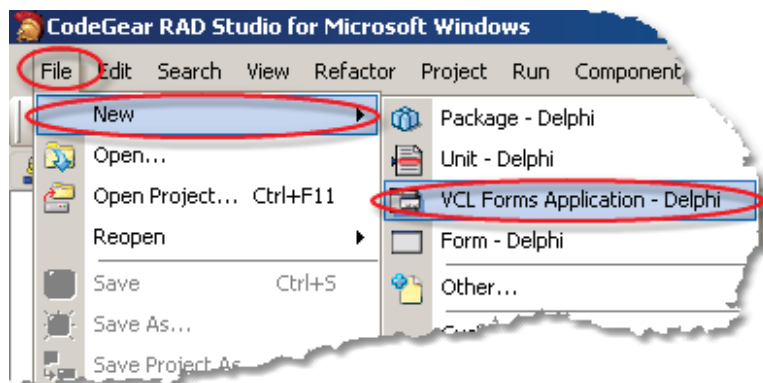
1. [Создание нового проекта](#) в среде CodeGear RAD Studio. Добавление к проекту дополнительных форм и фреймов. Далее на формах и на фреймах следует расположить компоненты, которые будут изображать мнемосхему. Компоненты могут быть как из стандартной поставки среды программирования CodeGear RAD Studio, так и созданные сторонними производителями. При большом количестве объектов на мнемосхеме и потребностью ее масштабирования рекомендуется воспользоваться [SVG графикой](#) для рисования мнемосхемы.

2. Определить с какими OPC серверами будет взаимодействовать приложение, т.е. к элементам каких серверов будет производиться привязка (динамизация) компонентов. Для настройки взаимодействия с OPC серверами введено понятие "псевдонима OPC сервера". [Псевдоним OPC сервера](#) отражает настройки соединения с сервером. Если взаимодействие осуществляется только с сервером "Dep.Model.x", то редактирование псевдонимов OPC серверов не требуется. Далее следует, исходя из иерархической структуры элементов OPC сервера определить, как будет происходить [имя образование привязки](#) к элементу OPC сервера и [привязать динамизируемые компоненты](#).

3. Используя встроенный редактор для задания привязок к элементам модели, можно реализовать большинство мнемосхем. Если же от приложения требуется более большой гибкости работы, то потребуются основные знания языка Delphi или C++ и библиотеки VCL. Написав минимум кода, используя уже созданные классы можно реализовать [дополнительные возможности](#), которые не заложены в редакторе состояний компонентов.

9.5.3.2 Создание проекта

Для создания нового проекта отображения мнемосхемы, требуется создать обычный проект (VCL Forms Application) в среде CodeGear RAD Studio (см. рис.) или использовать уже имеющийся проект, содержащий VCL-формы:



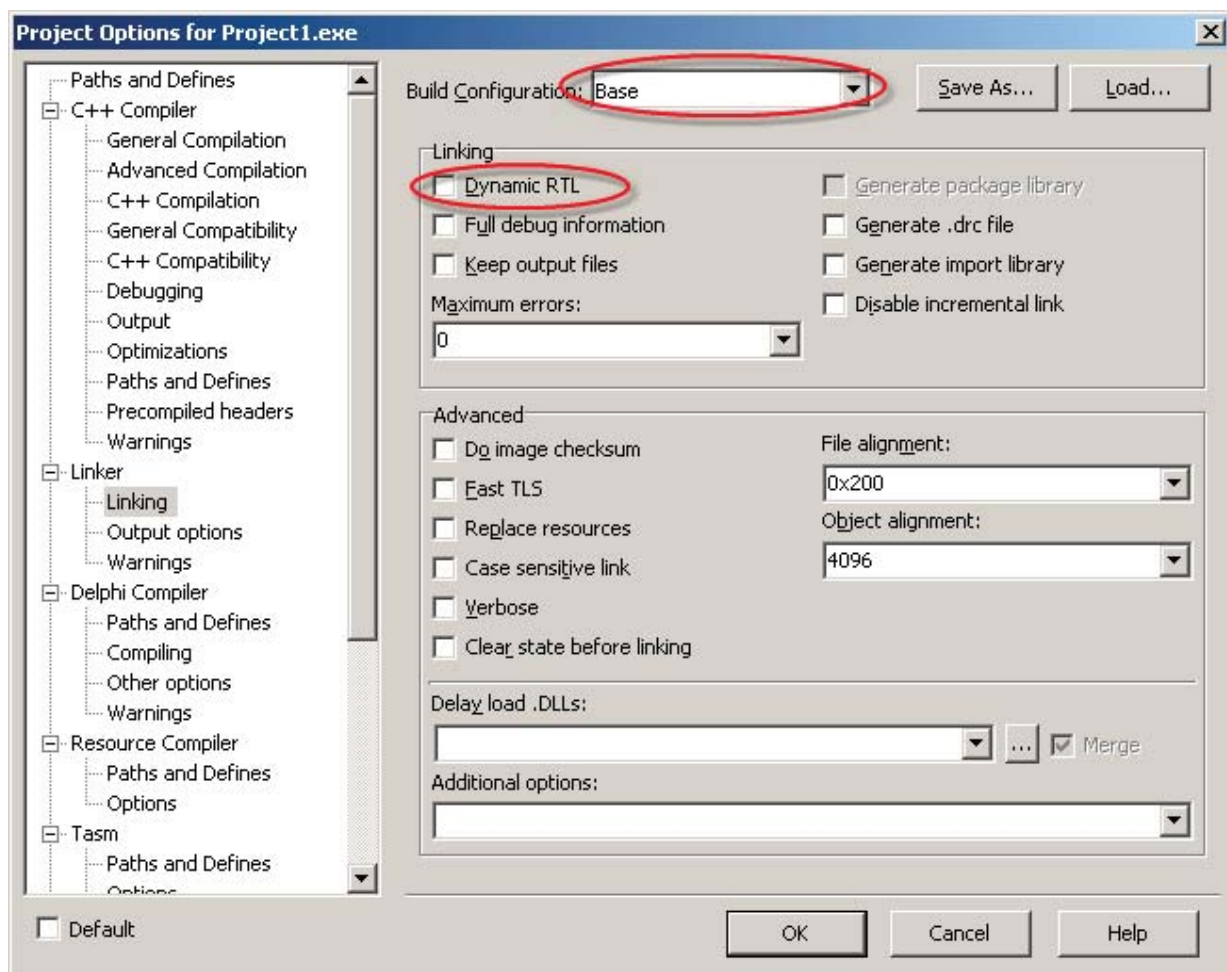
Для дальнейшей работы требуется [настроить проект](#) для поддержки создания динамизации компонентов.

9.5.3.3 Особенности сборки проекта для различных сред программирования

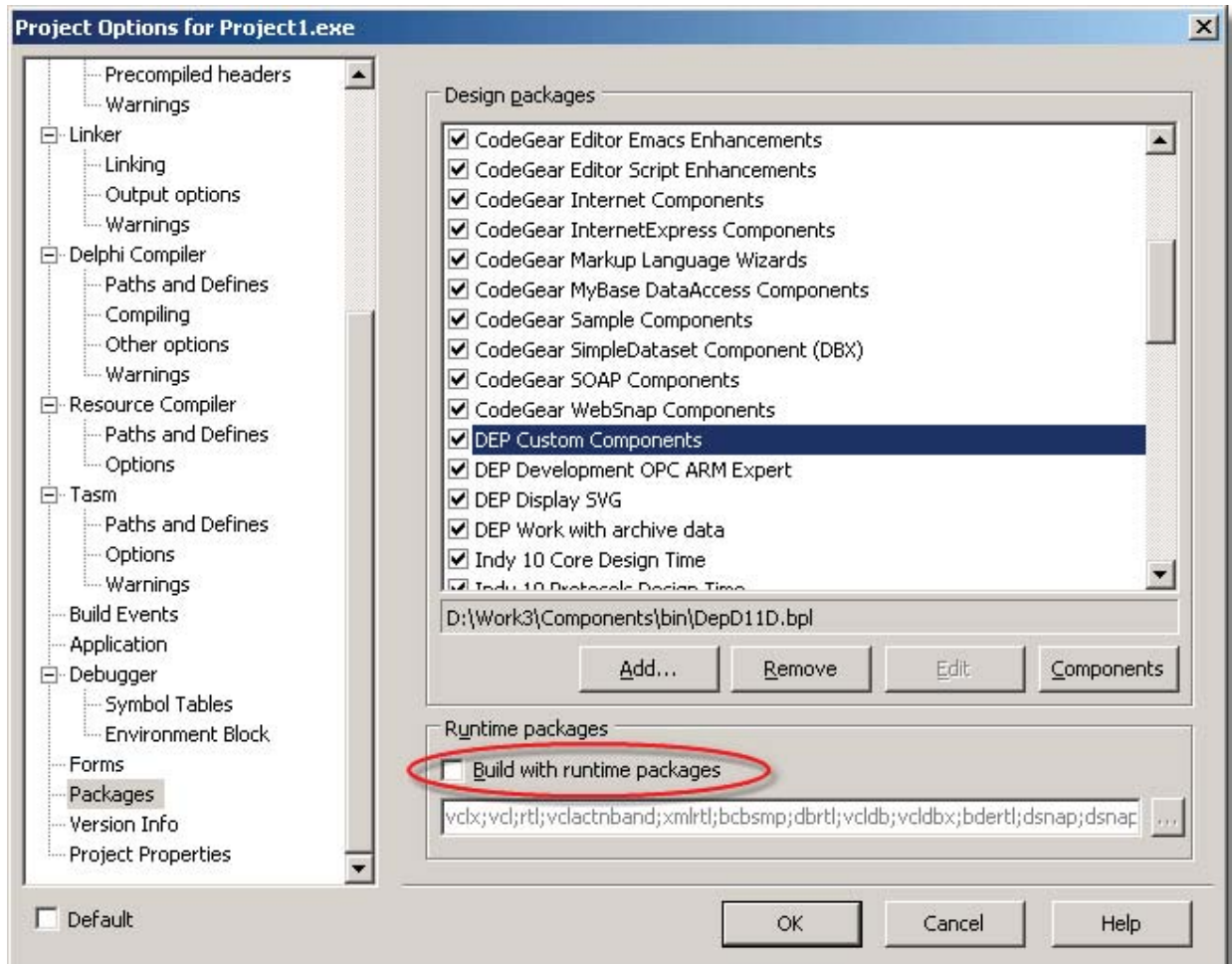
CodeGear RAD Studio 2007 (C++ Builder)

Для устранения ошибки (см. рис.), связанной с инициализацией менеджера памяти, при запуске приложения требуется отключить динамическое связывание с библиотеками времени выполнения. Это также позволит собрать приложение, которое не будет зависеть от библиотек, поставляемых со средой разработки.





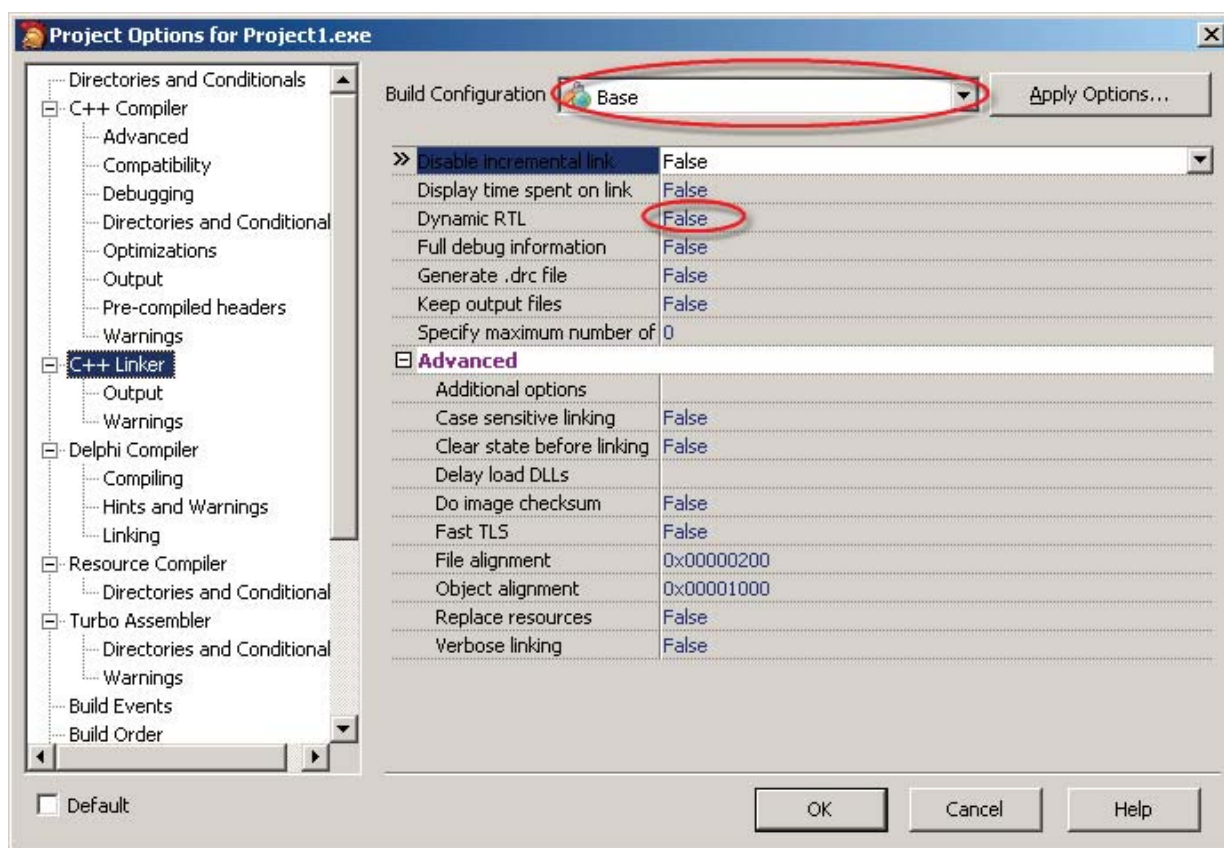
Рекомендуется также для статического включения кода компонентов отключить сборку с пакетами времени выполнения (см. рис. ниже).



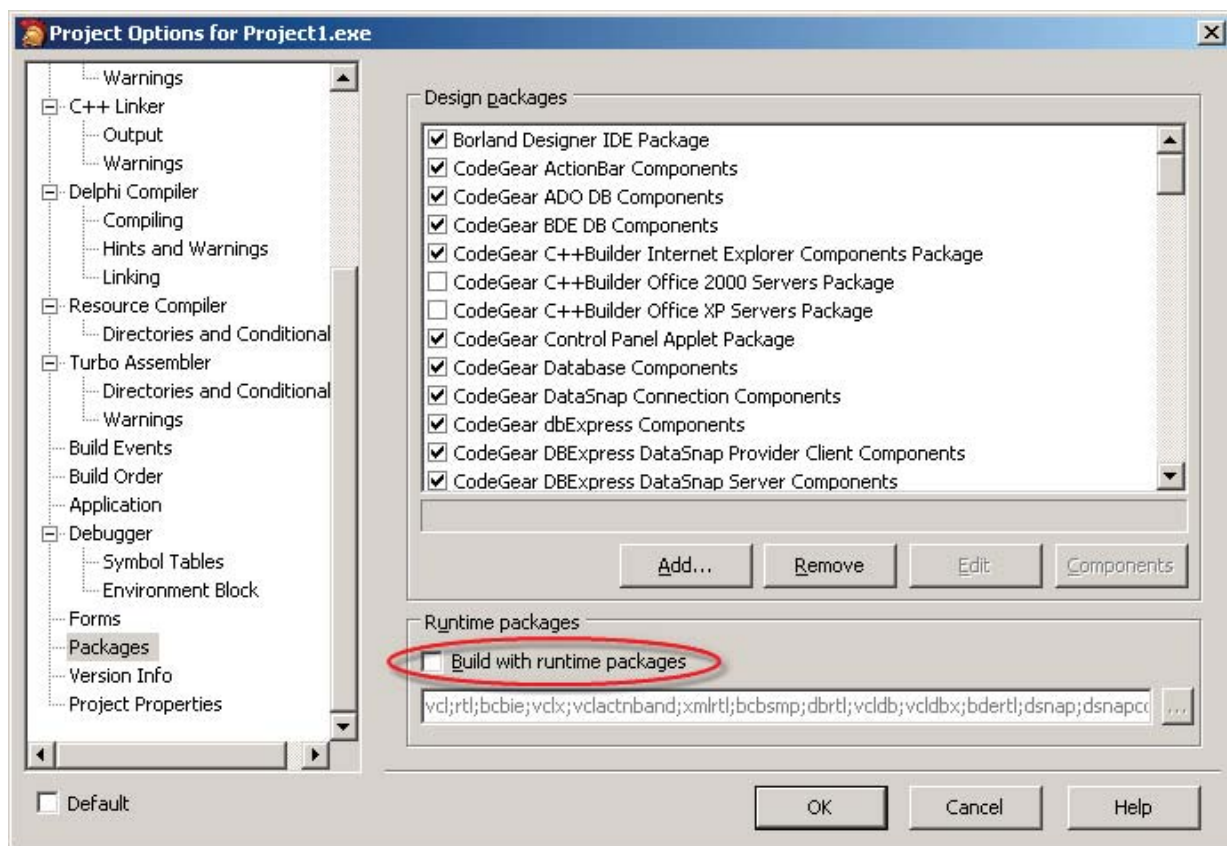
CodeGear RAD Studio 2009 (C++ Builder)

Внимание! Не используйте при сохранении проекта в пути к файлам русские названия папок. Сохраненный проект при последующем открытии может не открыться.

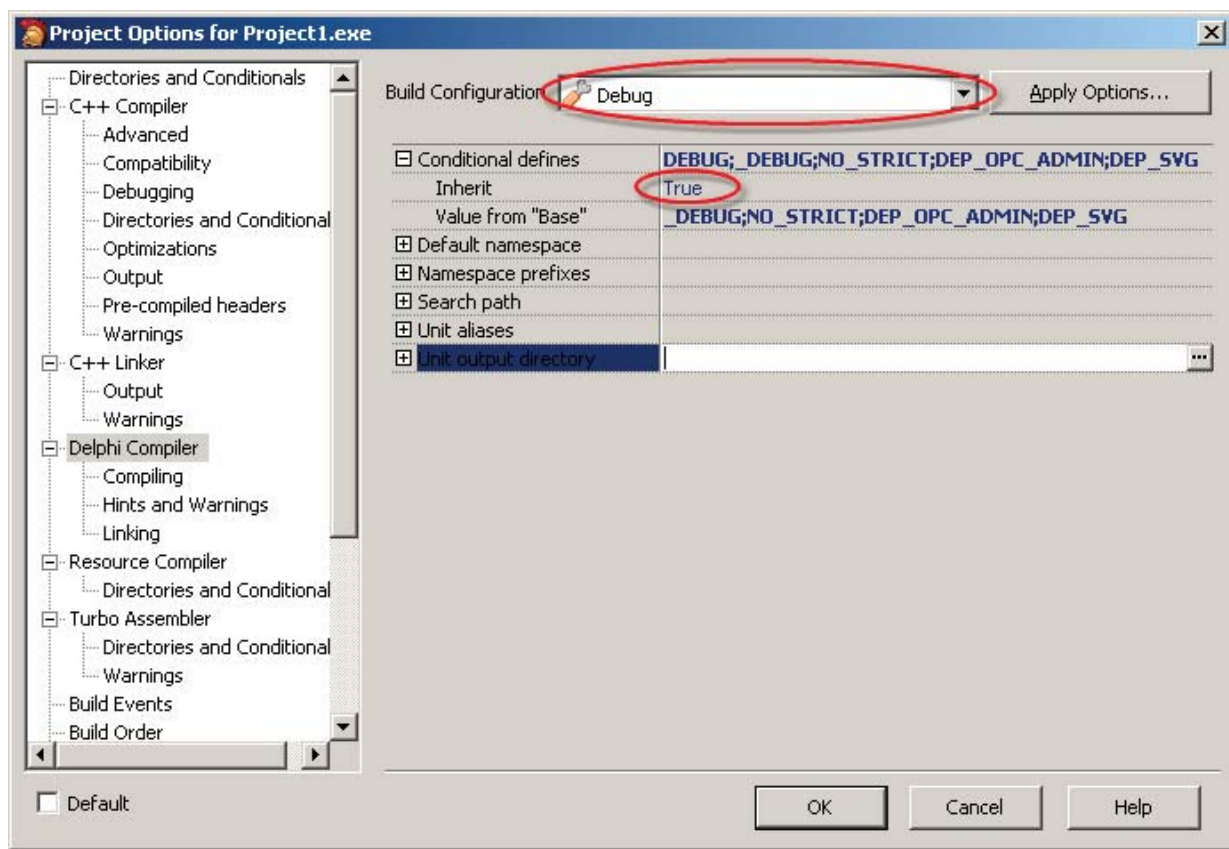
Для устранения ошибки (см. рис.), связанной с инициализацией менеджера памяти, при запуске приложения требуется отключить динамическое связывание с библиотеками времени выполнения. Это также позволит собрать приложение, которое не будет зависеть от библиотек, предоставляемых со средой разработки.



Рекомендуется также для статического включения кода компонентов отключить сборку с пакетами времени выполнения (см. рис. ниже).



Для поддержки SVG графики и работы с OPC пользователями требуется включить наследование переменных компилятора, если идет сборка конфигурации "Debug" (см. рис. ниже).



9.5.3.4 Настройки OPC проекта

9.5.3.4.1 Основные

Для вызова диалога настройки OPC проекта в среде CodeGear RAD Studio выберите "OPC Config..." в меню "DEP OPC Design".



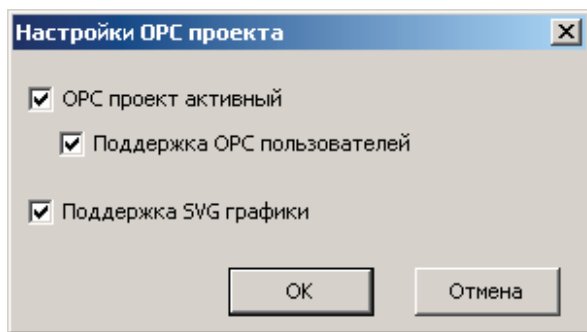
В появившемся окне:

OPC проект активный - поддержки OPC динамизации компонентов, как во время разработки так и во время выполнения программы. После установки данного флажка станут доступны разделы "OPC Design Form...", "OPC Design Group..." и "OPC Aliases..." меню "DEP OPC Design";

Поддержка OPC пользователей - добавляет возможность идентификации пользователя в создаваемом APM'е и задания

для него соответствующих прав доступа. Требуется чтобы OPC проект был активным(более подробно [Работа с пользователями](#));

Поддержка SVG графики - включает возможность отображения и динамизации SVG графики(более подробно [Поддержка SVG графики](#)).



После настройки опций OPC проекта рекомендуется, задать [псевдонимы OPC серверов](#), с которыми будет производиться работа, а также следует обратить внимание на [Особенности сборки проекта для различных сред программирования](#).

При изменении настроек OPC проекта требуется произвести полную сборку проекта.

9.5.3.4.2 Псевдонимы OPC серверов

Псевдонимы OPC серверов служат для описания настроек взаимодействия с ними и дают возможность работы с несколькими OPC серверами одновременно.

Для вызова диалога настройки OPC псевдонимов в среде CodeGear RAD Studio выберите "OPC Aliases..." в меню "DEP OPC Design".



В появившемся окне можно добавить, удалить, настроить выбранный псевдоним.

Для OPC псевдонима следует определить следующие поля:

- **Имя псевдонима.** Имя псевдонима, которое в дальнейшем будет использоваться для задания привязки к элементам OPC сервера. Рекомендуется определить это имя один раз (перед привязкой компонентов), т.к. оно будет использоваться в имени привязки (см. [Принципы имя образования для привязок](#)), если только этот псевдоним не задан как псевдоним по умолчанию.
- **Название OPC сервера.** Имя сервера, под которым он зарегистрирован в операционной системе. Имя сервера можно узнать у разработчика этого сервера. В выпадающем списке можно выбрать только имена локально зарегистрированных серверов.
- **Удаленный сервер.** Если данный флажок установлен, то требуется задать имя компьютера или его IP адрес, на котором запущен OPC сервер. В этом случае взаимодействие с сервером будет осуществляться по сети и для корректной работы потребуется настроить DCOM (Настройка для Win2000 и WinXP SP2).
- **Разделитель в OPC имени.** Символ, использующийся в имя образования привязок. (см. [Принципы имя образования для](#)

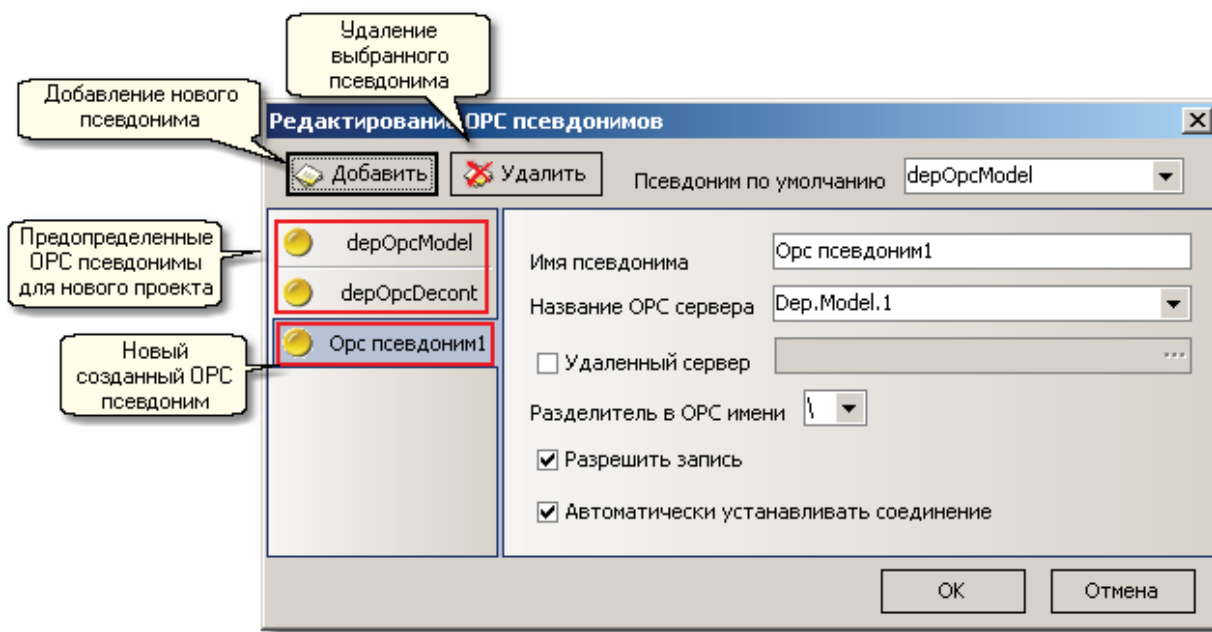
[привязок](#)).

- **Разрешить запись.** Если флажок установлен, то разрешена запись в элементы OPC сервера, иначе запись будет блокироваться. Блокировка записи производится не самим OPC сервером, а создаваемым приложением, в отличии от настроек прав активного пользователя, если заданы [пользователи](#) для разрабатываемого приложения.
- **Автоматически устанавливать соединение.** Указывает как будет происходить соединение с OPC сервером. Если флажок установлен, то соединение с сервером будет происходить автоматически при старте приложения, иначе соединению потребуются установить программно.

Для создаваемого OPC приложения определены два псевдонима:

- **depOpcModel.** Разработка модели OPC-сервера (OPC-модели) ведётся в [Конструкторе OPC модели](#). Для более удобной разработки проекта отображения рекомендуется сначала создать OPC-модель.
- **depOPCDecont.** Представляет собой OPC сервер базы параметров контроллера WinDecont. (см. описание WinDecont).

Из созданных OPC псевдонимов должен быть определен псевдоним по умолчанию, для которого не требуется явно указывать его псевдоним в OPC привязке.



9.5.3.4.3 Принципы имя образования для привязок

Привязка - символьная строка, указывающая с каким элементом OPC сервера будут сопоставлены какие либо действия.

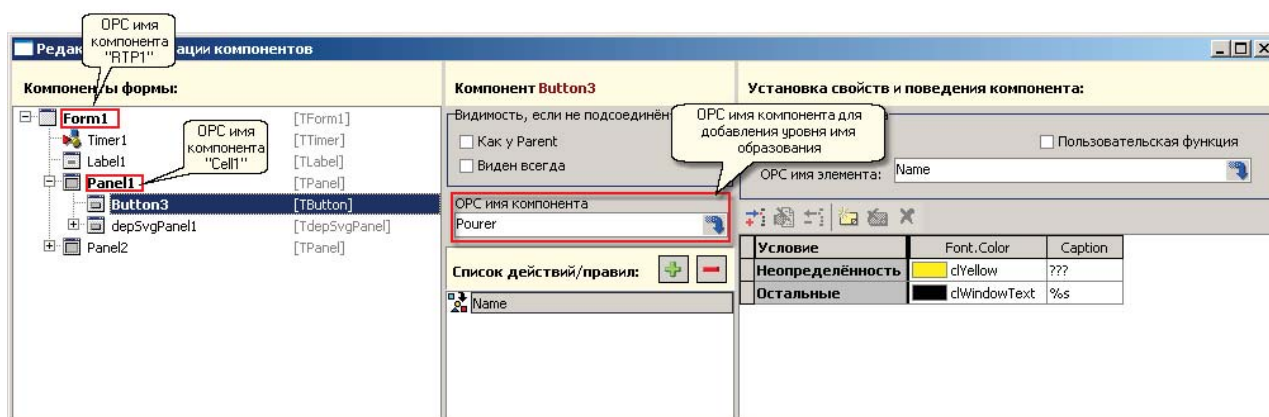
Формат привязки: <[Имя псевдонима]><Разделитель><Имя привязки>.

Имя псевдонима указывает с каким OPC сервером будет происходить соединение. Если используется псевдоним по умолчанию, то указание имени псевдонима необязательно.

Иерархическая структура элементов в OPC серверах позволяет нам в какой-то мере отразить её на создаваемой мнемосхеме. Данный принцип основывается на способе имя образования привязок, которые создаются в [редакторе динамизации компонентов](#). Возможно два способа имя образования:

- **Абсолютное.** Имя привязки начинающаяся на символ "\", считается абсолютной. При этом в процессе выполнения программы в OPC сервере запрашивается элемент с именем указанным после символа "\";
- **Относительное.** OPC имя привязки перед запросом соответствующего элемента из OPC сервера вычисляется в соответствии с заданными уровнями имя образования. Уровень имя образования может быть задан для любого компонента.

OPC имя привязки вычисляется сложением всех OPC имен компонентов по Parent`у, OPC имени конечного компонента и OPC имени элемента. При сложении между именами вставляется символ разделения, заданный в [псевдониме OPC сервера](#). Т.е. для привязки к компоненту *Button3* полное OPC имя будет *RTP1|Cell1|Pourer|Name*. При относительном имени возможно также обращаться к нижележащим уровням, используя набор символов *..\|* т.е. если нам требуется в *Button3* создать привязку к *RTP1|Name*, нужно в OPC имени элемента написать *..\|..|Name*.



9.5.3.5 Редактор динамизации компонентов

Для вызова редактора динамизации компонентов на любом из расположенных на форме компонентов кликните правой клавишей мыши и выберите пункт меню *OPC Design Form...* или нажмите комбинацию клавиш *<Shift+F3>*.

Дерево компонентов построено по принципу: "кто *Parent* компонента, тот и его владелец", для SVG объектов под *Parent`*ом подразумевается SVG группа.

Здесь и далее условимся VCL компонент или SVG объект называть просто *объектом* для простоты изложения.

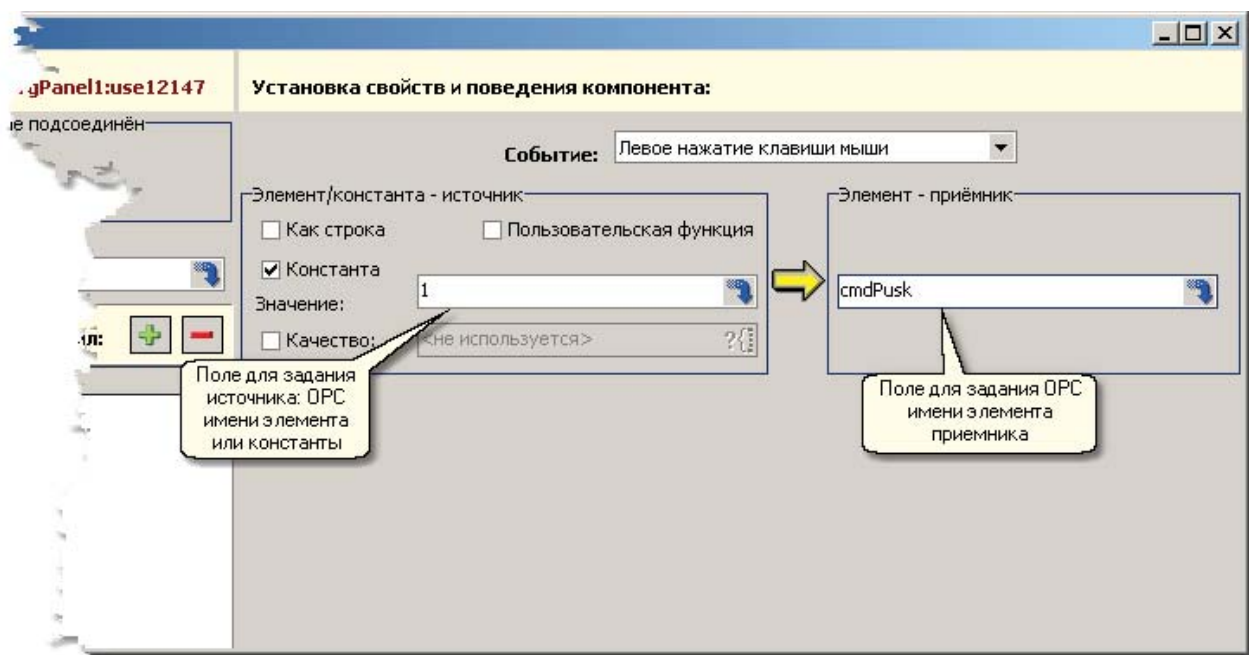
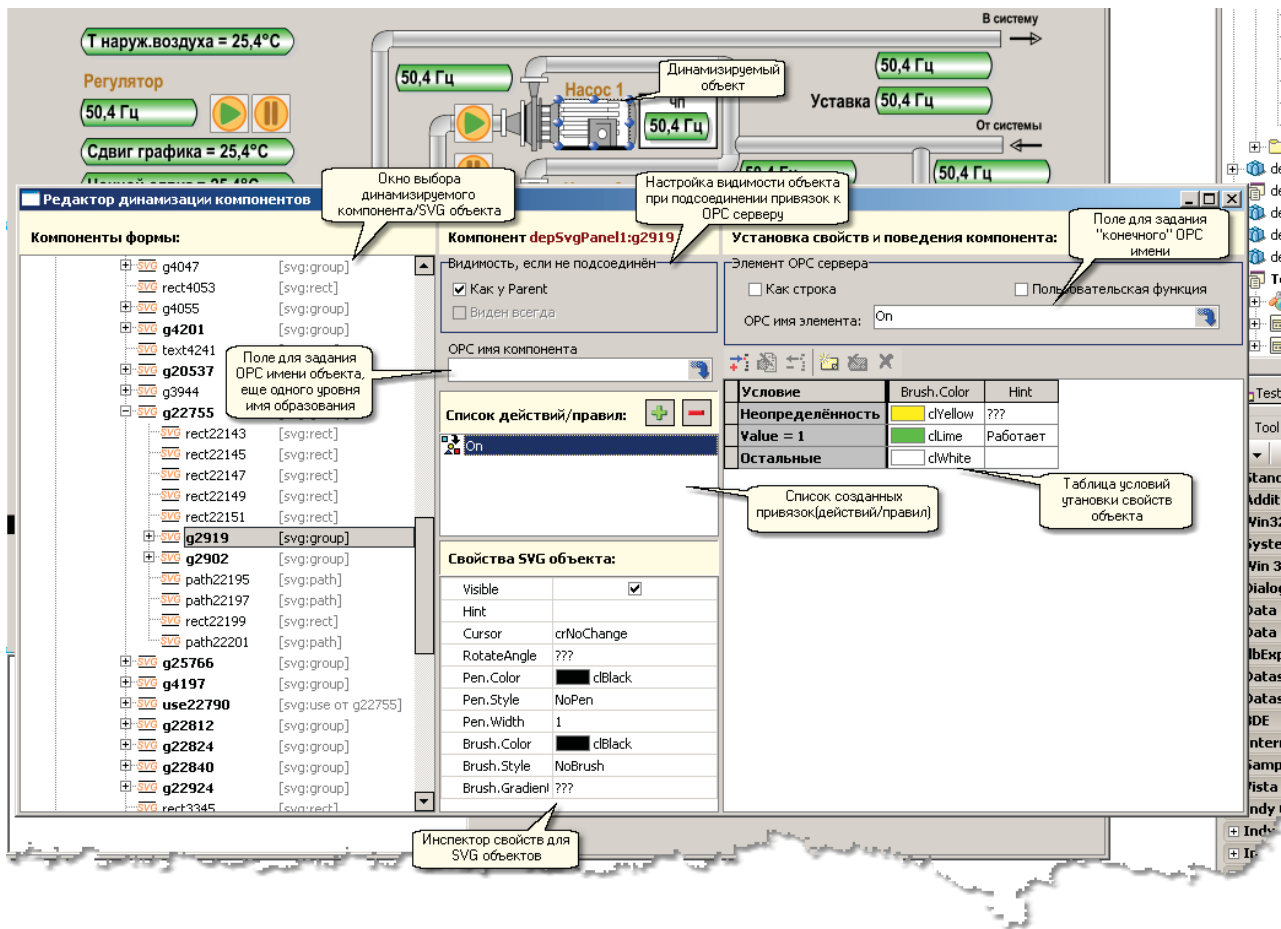
Выбрав в дереве объект, можно для него создать привязки к элементам OPC сервера для этого нужно нажать кнопку с зеленым плюсом. В результате этого появится окно [выбора свойства или действия](#), которое должно совершиться при выполнении заданного условия. После выбора свойства или действия будет создана привязка к данному объекту, для которой потребуется возможно [задать состояния сравнения](#) и [привязки к свойствам объекта](#). При привязки к событиям потребуется задать OPC элемент приемник для записи и OPC элемент источник или константу.

Для визуальных объектов есть возможность управлять их видимостью в зависимости от наличия заданного одного или нескольких OPC элементов в сервере:

Как у Parent - если флажок установлен, то настройки видимости будут братья из *Parent`*а данного объекта, в противном случае будет анализироваться флажок *Виден всегда*.

Виден всегда - если флажок установлен, то объект будет виден независимо от наличия OPC элемента на сервере, в противном случае объект будет скрыт, если хотя бы один OPC элемент отсутствует на сервере.

OPC имя привязки(компонента) можно ввести вручную в соответствующем поле или, нажав кнопку  для вызова диалога [выбора OPC имени привязки](#).



9.5.3.5.1 Привязка к свойствам объекта

Для любого VCL компонента и SVG объекта могут быть динамизированы:

- любые его [публичные свойства](#) (свойства, которые показывает Object Inspector в среде программирования CodeGear RAD Studio), для SVG объекта набор основных свойств (цвет и стиль заливки, обводки, хинт, курсор, видимость, для текста - шрифт, содержание);
- [события нажатия клавиш мыши](#) (левое нажатие, правое нажатие, двойное нажатие левой клавиши мыши, двойное нажатие правой клавиши мыши) для записи значений в элементы OPC сервера;
- [специальные свойства](#) (мигание).

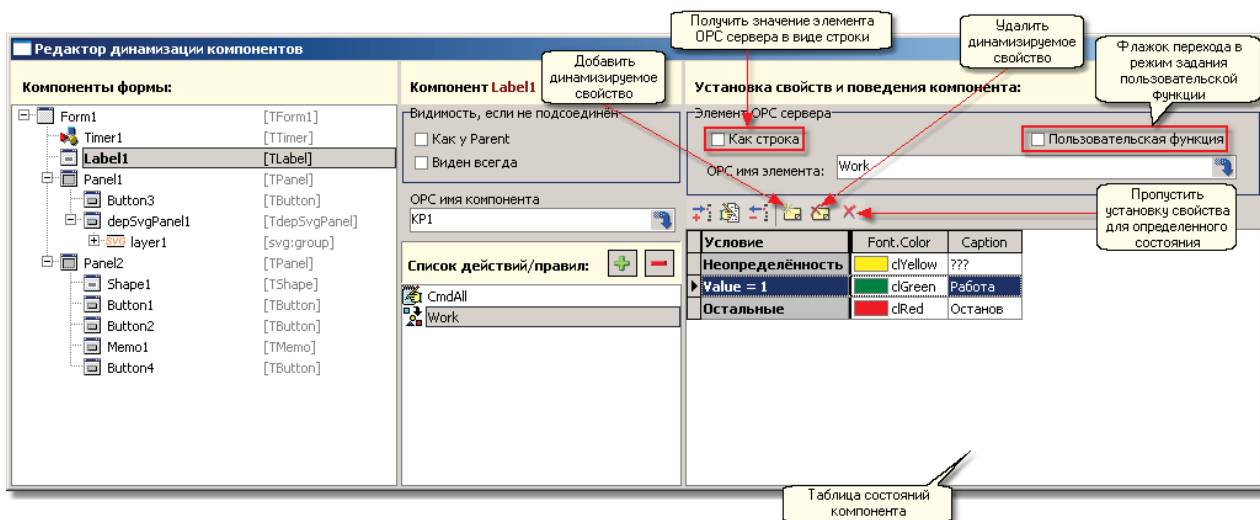
9.5.3.5.1.1 Привязка к публичным свойствам

Добавление и удаление динамизируемых свойств для объекта происходит с помощью соответствующих кнопок, описанных на рисунке. При этом создается столбец с заголовком выбранного свойства. В ячейках данного столбца для каждого [состояния сравнения](#) можно задать свое значение свойства. Каждое значение свойства редактируется в соответствии с его базовым типом.

Для свойств с базовым типом UString(LString), Float и Integer предусмотрено форматирование значений. Форматирование значений аналогично работе функции Format или FormatDateTime (для типа TDateTime). Например, если динамизируется свойство Caption у компонента TLabel, то в строке столбца "Caption" можно записать "ПУ %d". В результате если будет установлено данное [состояние при сравнении](#), то вместо %d, будет выведено значение (Value) элемента модели. В отличии от функции Format использование маски "%s" позволяет вывести любое значение вне зависимости от типа Value элемента модели.

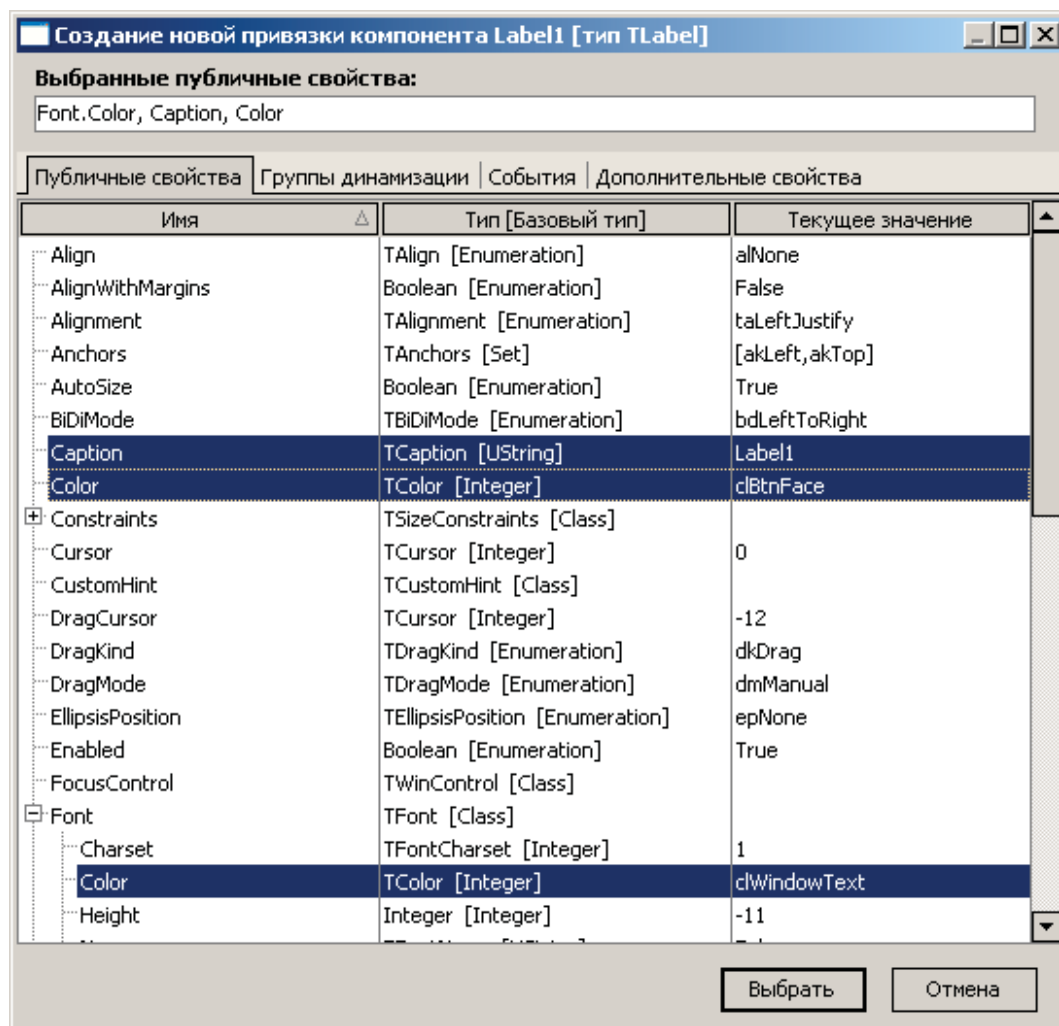
Для получения элемента OPC сервера в виде строки установите галочку "Как строка". В данном случае если со значением элемента в OPC сервере сопоставлена строка, то Value примет строковое значение.

Существует возможность вычислять значения Value и Quality, которые передаются для сравнения на основе реальных значений из OPC сервера, в [пользовательской функции](#).



При добавлении новой привязки в [редакторе состояний компонентов](#) или при редактировании привязки можно выбрать свойства компонента для динамизации. Возможно выбрать сразу нескольких свойств, для этого при выделении каждого свойства удерживайте клавишу **Ctrl**.

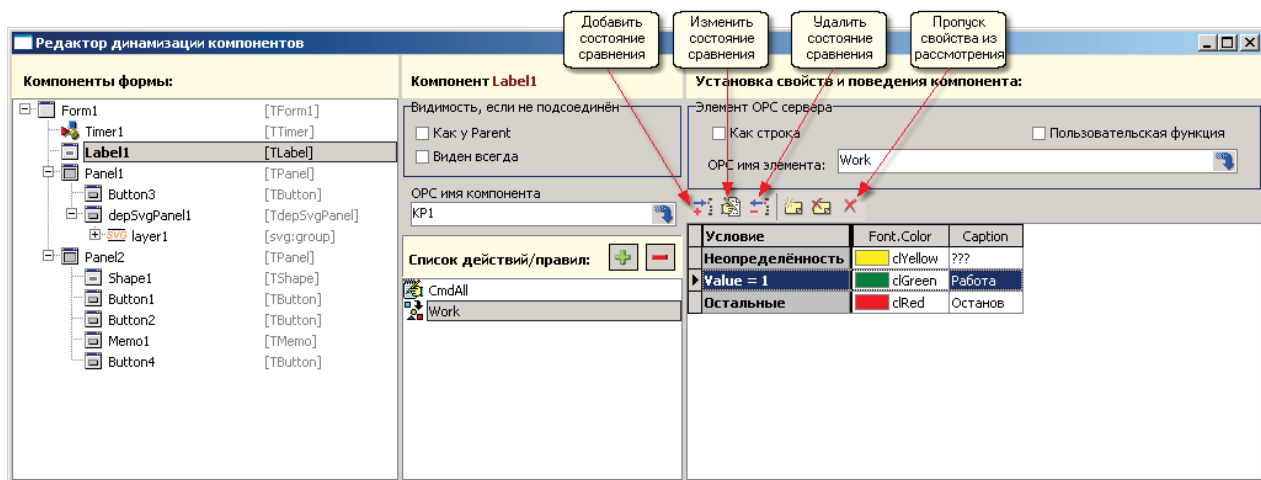
Описание свойств VCL компонентов можно прочитать в справке к CodeGear RAD Studio.



9.5.3.5.1.2 Добавление состояний сравнения

Для задания поведения компонента в зависимости от значения элемента модели требуется создать состояния сравнения (условия). При каждом изменении значения элемента модели (Value или Quality) происходит поиск сверху вниз истинного состояния. При его нахождении происходит установка соответствующих динамизируемых свойств в выбранной строке.

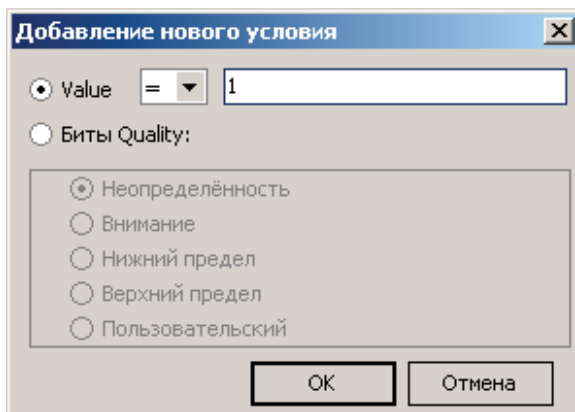
Для добавления, изменения и удаления состояний сравнения можно воспользоваться соответствующими кнопками, обозначенными на рисунке.



Добавление или изменении состояния производится в окне "Добавление нового условия" ("Изменение условия"). В этом окне требуется задать условие сравнения с значением Value (возможно использование знаков "=" (равно), "!=" (не равно), "<" (меньше), ">" (больше)) или с каким либо битом Quality (выбор какого-либо бита Quality означает проверку установки этого бита в Quality изменившегося элемента OPC сервера).

При изменении значения привязанного элемента модели (изменилось его Value или Quality) в процессе выполнения программы происходит поиск сверху вниз истинного выражения. Обязательное состояние "Остальные" (всегда последнее), которое нельзя удалить из рассмотрения. При найденном истинном утверждении происходит установка всех свойств описанных в данной строке.

Существует возможность пропустить свойство из рассмотрения в определённой строке. При этом если в найденном истинном утверждении какое-либо свойство пропущено из рассмотрения, поиск по условиям продолжится для установки пропущенных свойств.

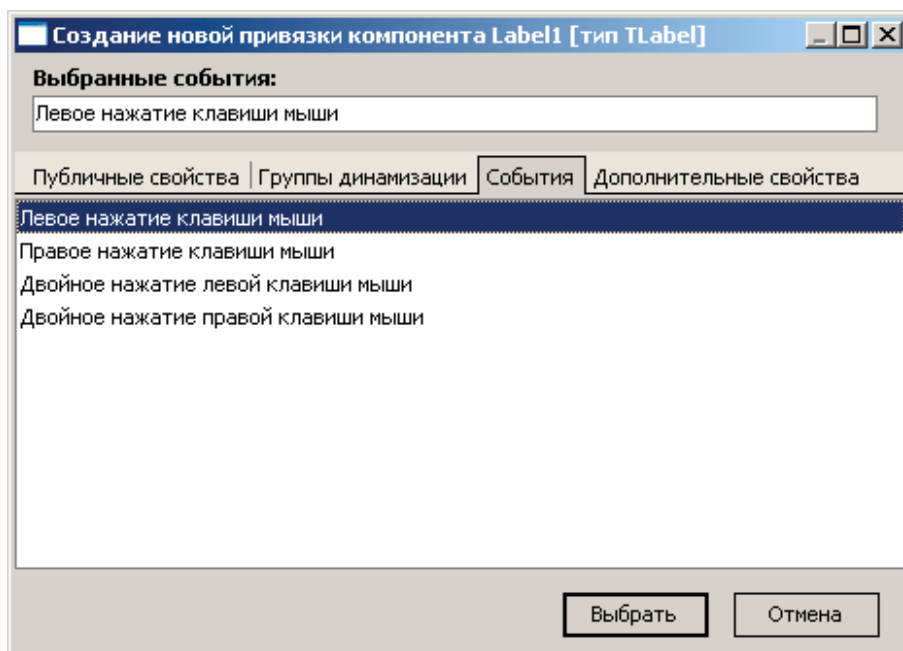


9.5.3.5.1.3 Привязка к событиям для записи

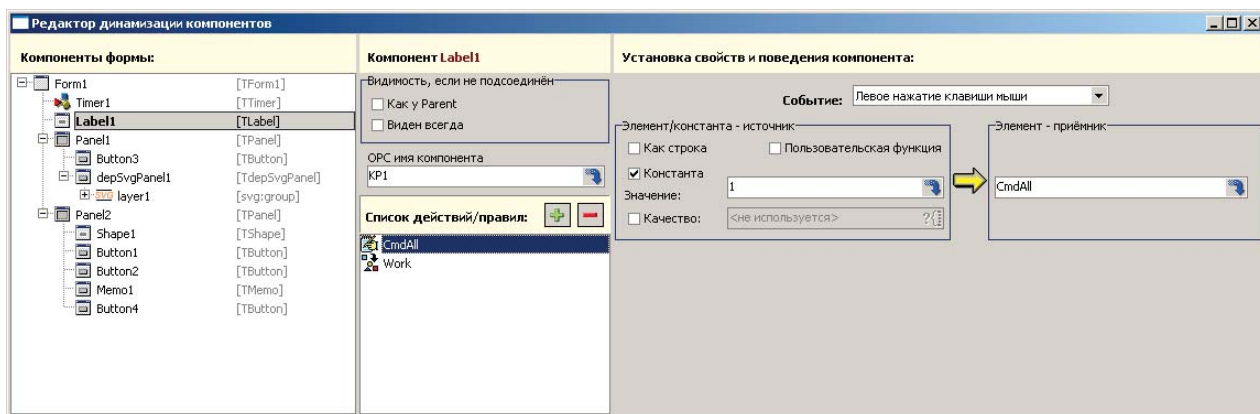
Для визуальных компонентов возможна привязка к действиям пользователя. В данной версии доступны четыре возможные события:

- левое нажатие клавиши мыши;
- правое нажатие клавиши мыши;
- двойное нажатие левой клавиши мыши;
- двойное нажатие правой клавиши мыши;

По данным событиям возможна запись значения в элемент модели, а также вызов пользовательской функции перед записью. Если компонент невидим, то данные события не могут произойти.

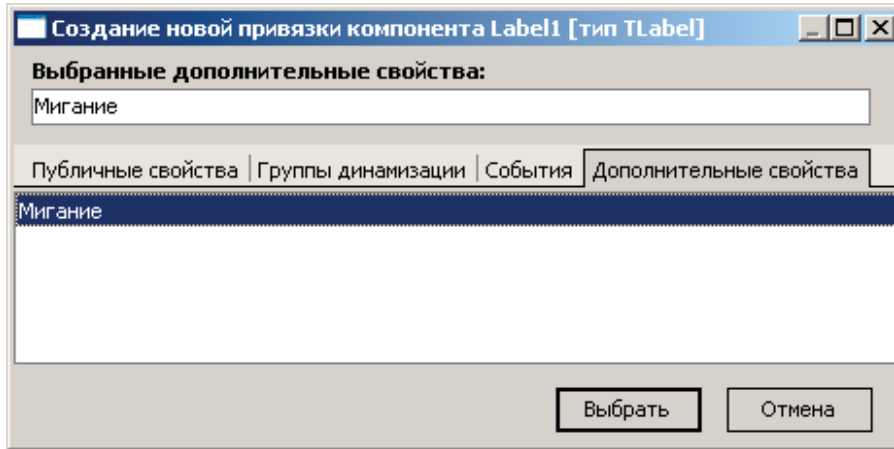


При редактировании привязки к событиям для записи требуется указать OPC элемент-приемник и OPC элемент-источник или константу. Возможен вызов [пользовательской функции](#) для источника, в которой можно отменить запись, изменить записываемое значение.



9.5.3.5.1.4 Привязка к специальным свойствам

В данной версии возможна привязка только к свойству "Мигание" (только для визуальных компонентов). Под миганием компонента понимается периодичное изменение набора его свойств.

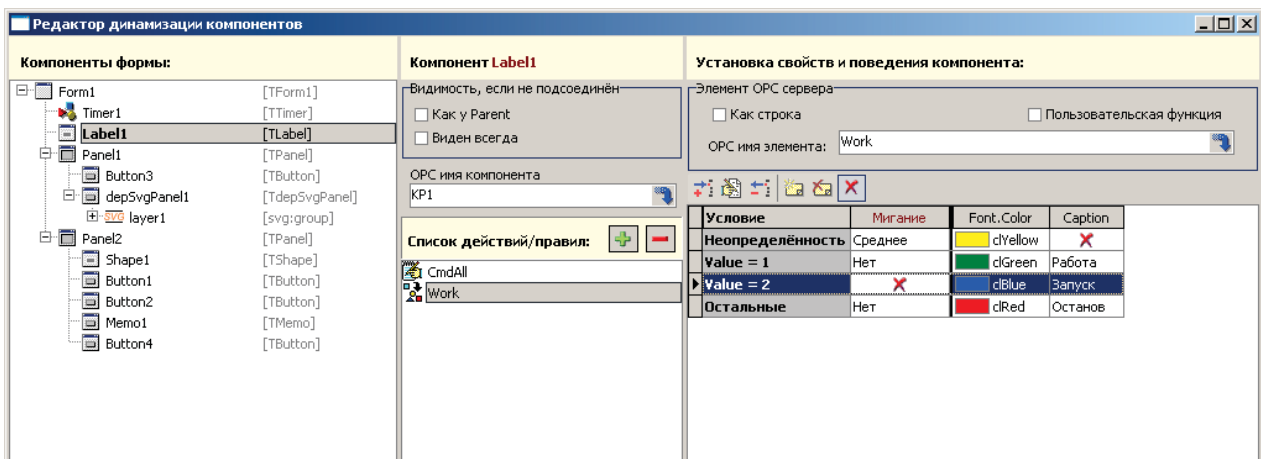


Специальное свойство - мигание позволяет установить поочередное изменение набора публичных свойств объекта во времени. Свойство мигание может быть добавлено в любой момент создания [привязки к публичным свойствам](#). Столбец "Мигание" будет всегда установлен первым в наборе столбцов свойств. Возможны четыре значения свойства "Мигание":

- "Нет" - не мигать;
- "Медленное" - медленная скорость мигания;
- "Среднее" - средняя скорость мигания;
- "Быстрое" - высокая скорость мигания.

При установлении состояния "мигать" (не в состоянии "Нет") компонент переходит в режим мигания. В этом режиме происходит поочередная смена свойств: 1/3 цикла компонент в состоянии "Мигание" - устанавливаются все значения свойств для установившегося состояния; 2/3 цикла компонент - в основном состоянии. В основном состоянии, проходя по всем привязкам, устанавливаются лишь те свойства, которые удовлетворяют условиям и в которых свойство "Мигание" установлено в "Нет".

Например, для Label1, если элемент OPC сервера станет неопределенным, то компонент перейдет в состояние "мигания". При этом мигать он будет в желтый цвет из зеленого или красного в зависимости от того какое значение принимало Value до неопределенности. Caption метки при мигании не будет меняться, т.к. это свойство пропущено в состоянии "мигать". Состояние "Value = 2" никак не участвует в мигании, т.к. для мигания оно является пропущенным.



9.5.3.5.2 Выбор OPC имени привязки



Данный диалог предназначен для выбора OPC имени привязки. Для каждого [OPC псевдонима](#) в проекте создается своя закладка с иерархическим представлением OPC элементов. Рисунок в виде кружка на каждой закладке показывает состояние соединения с OPC сервером:







- соединение с OPC сервером установлено;



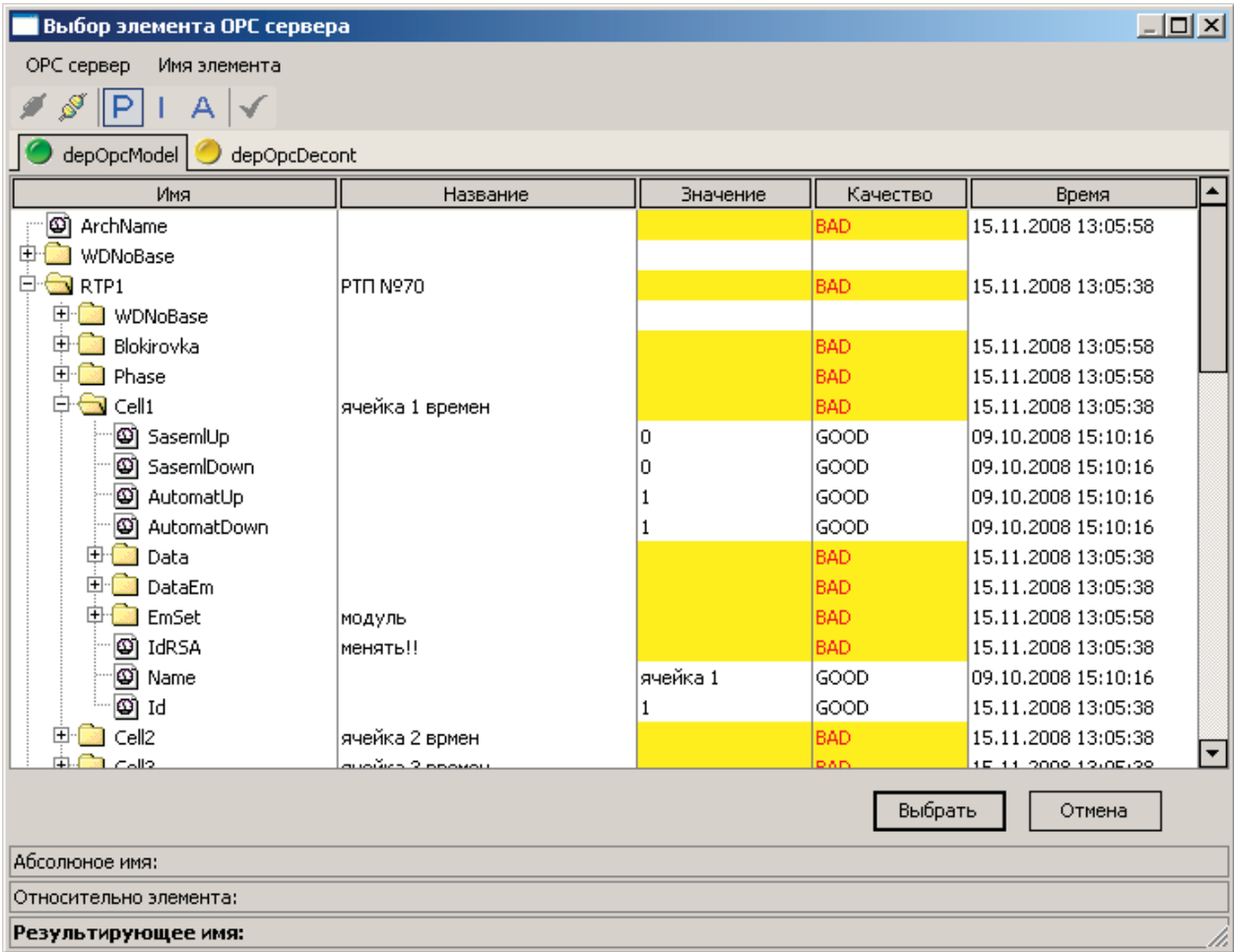
- соединение с OPC сервером по каким-либо причинам не удается установить, нужно проверить настройки [OPC псевдонима](#) и доступность OPC сервера.

Соединение с OPC сервером устанавливается автоматически, если в OPC псевдониме указано *автоматически устанавливать соединение*, иначе соединение потребуется устанавливать вручную, нажав кнопку . Для отсоединения потребуется нажать кнопку .

Выбор элемента OPC сервера можно производить в трех режимах, переключаясь соответствующими кнопками:



-  Авто имя относительно Parent. Из имени будет урезана начальная часть, составляющая OPC-имя Parent`а элемента.
-  Имя относительно заданного элемента. Выбрав этот режим требуется также указать, нажав кнопку , OPC элемент, относительно которого будет производиться имя образование.
-  Абсолютное (полное) имя элемента.

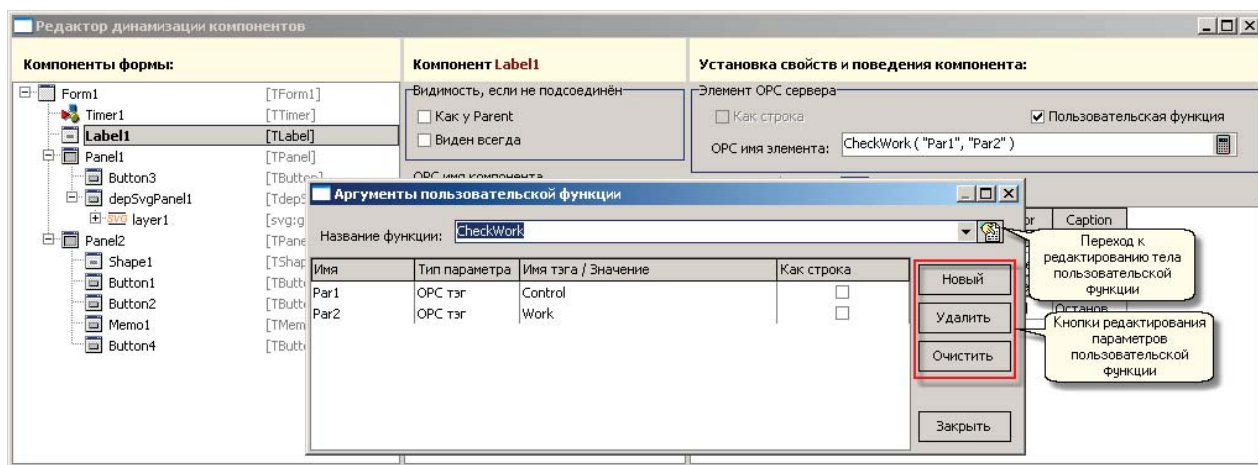
Все перечисленные действия можно произвести также выбрав соответствующий пункт меню или по "горячим" клавишам.



9.5.3.5.3 Пользовательские функции

Пользовательские функции позволяют проанализировать значения OPC тэгов до выполнения состояний сравнения или до выполнения записи в OPC тэг. Пользовательскую функцию можно задать как для [привязки к публичным свойствам](#) компонента, так и в [привязке к событиям для записи](#). При написании кода пользовательской функции, требуется подключить заголовочный файл `depOpc.hpp` для C++Builder, для Delphi в секцию `uses` добавить подключение модуля `depOpc`.

При установке флажка пользовательской функции кнопка  в поле выбора элемента OPC сервера заменяется на кнопку  редактирования вызова редактора пользовательской функции.



В редакторе аргументов пользовательской функции возможно добавление, удаление и очистка всех аргументов. Функция будет вызываться каждый раз, когда происходит инициализация или изменение(меняется Value или Quality OPC элемента) хотя бы одного параметра с типом OPC тэг. Для каждого аргумента указывается:

- **Имя** - название параметра, по которому можно к данному аргументу обращаться в коде пользовательской функции;
- **Тип параметра** может принимать два значения: *OPC тэг*(в следующем поле указывается относительное или абсолютное название элемента OPC сервера) и *Константа*(в следующем поле указывается строковая константа);
- **Имя тэга / Значение** - название элемента OPC сервера или константа в зависимости от поля Тип параметра;
- **Как строка** - получать значение OPC элемента в виде строки(только для типа параметра - *OPC тэг*)

Возможно создание двух типов пользовательских функций:

- TdepOpcUserSetFuncNotify - создается в [привязках к публичным свойствам](#);
- TdepOpcUserWriteFuncNotify - создается в [привязках к событиям для записи](#).

```
TdepOpcUserSetFuncNotify = procedure (const aFullRelativeOpcName: string; aListParUserFunc: TdepOpcList;
aIndexParChange: Integer; aAllConnected: Boolean; aAllChanged: Boolean;
var aValue: Variant; var aQuality: TdepOpcQuality; var aSetProperties: Boolean) of object;
```

```
TdepOpcUserWriteFuncNotify = procedure (const aFullRelativeOpcName: string; aListParUserFunc: TdepOpcList;
aIndexParChange: Integer; aAllConnected: Boolean; aAllChanged: Boolean;
var aWriteValue: Variant; var aQualitySpecified: Boolean; var aWriteQuality: TdepOpcQuality;
var aWriting: Boolean) of object;
```

Описание аргументов:

aFullRelativeOpcName - полное имя привязки, полученное в результате имя образования;

aListParUserFunc - список объектов класса TdepOpcParUserFunc, в котором находится описание параметров, заданных в редакторе аргументов пользовательской функции, и их текущие значения. Список представляет собой типизированный список класса TList;

aIndexParChange - индекс параметра в списке aListParUserFunc, для которого произошло изменение значения(Value или Quality).

aAllConnected - флаг подключения(наличия) всех параметров функции с типом OPC тэг в OPC сервере.

aAllChanged - флаг установления значения для всех параметров функции с типом OPC тэг в OPC сервере.

только для TdepOpcUserSetFuncNotify:

aValue - значение Value, выходной параметр, который будет анализироваться в [привязке к публичным свойствам](#) в соответствии с условиями состояний;

aQuality - значение Quality, выходной параметр, который будет анализироваться в [привязке к публичным свойствам](#) в соответствии с условиями состояний;

aSetProperties - выходной параметр, указывающий требуется(True) или нет(False) после вызова функции анализировать значения Value и Quality в [привязке к публичным свойствам](#) в соответствии с условиями состояний;

только для TdepOpcUserWriteFuncNotify:

aWriteValue - значение Value для записи в элемент OPC сервера;

aQualitySpecified - флаг, указывающий требуется(True) или нет(False) записывать значение Quality в элемент OPC сервера;

aWriteQuality - значение Quality для записи в элемент OPC сервера;

aWriting - флаг, указывающий требуется(True) или нет(False) производить запись Value и Quality;

Описание класса, используемого для элементов списка TdepOpcListParUserFunc:

```
TdepOpcParUserFunc = class
public
    property Name: string read fName; //название параметра, заданное в редакторе аргументов пользователь
    property ConstOrOpcName: string read fConstOrOpcName; //константа (TypeSource = tsConst) в виде строки
    property TypeSource: TdepOpcTypeSourceUserFunc read fTypeSource; //указывает, что хранится в поле Con
    property IsAsString: Boolean read fIsAsString; //брать значение элемента OPC сервера в виде строки(то
    property Value: Variant read fValue; //значение элемента OPC сервера(только для TypeSource = tsOpcTag
    property Quality: TdepOpcQuality read fQuality; //качество элемента OPC сервера(только для TypeSource
    property Connected: Boolean read fConnected; //флаг подсоединения элемента OPC сервера(только для Tur
end;
```

9.5.4 Программный доступ для расширенных возможностей

[Вызов редактора динамизации компонентов](#)

[Подключение динамизированных форм, фреймов и SVG-панелей для отображения их динамизации](#)

[Задание OPC имени динамизируемым объектам](#)

[Блокировка и разблокировка задания OPC имени](#)

[Синхронное чтение и запись значений в тэги\(элементы\) OPC сервера](#)

[Асинхронное чтение и запись значений в тэги\(элементы\) OPC сервера\(пользовательские привязки\)](#)

[Тип TdepOpcQuality и функции для работы с ним](#)

В файле depOpc.pas определен класс *TdepOpcApplication*. При запуске OPC приложения создается объект *gOpcApplication* этого класса. Практически весь программный доступ осуществляется через свойства и методы данного класса. Рекомендуется использовать только документированные свойства и методы. Весь программный код будет приведен на языке программирования Delphi.

При написании кода для Delphi требуется в секцию `uses` добавить подключение модуля `depOpc`, для C++Builder подключить заголовочный файл `depOpc.hpp`

Вызов редактора динамизации компонентов

```
procedure TdepOpcApplication.OpcDesignEdit(aWindow: TScrollingWinControl);
```

В процедуру в качестве аргумента *aWindow* требуется передать указатель на форму или фрейм, для которой будет редактироваться динамизация.

Подключение динамизированных форм, фреймов и SVG-панелей для отображения их динамизации

```
procedure TdepOpcApplication.AddFormOrFrame(aScrFormOrFrame: TScrollingWinControl;  
  aIsSetOpcName: Boolean = False; const aOpcName: string = '');
```

```
procedure TdepOpcApplication.AddSvgPanel(const aSvgPanel: TdepSvgPanel);
```

Перед использованием данных процедур для созданной формы, фрейма или SVG-панели требуется задать *Parent*(компонент его отображающий).

Процедура *AddFormOrFrame* - подключение формы или фрейма, ее параметры:

aScrFormOrFrame - указатель на добавляемую форму или фрейм;

aIsSetOpcName - флаг, указывающий требуется ли для подключенного компонента установить другое OPC имя, переданное в параметре *aOpcName*;

aOpcName - новое OPC имя, которое будет установлено после подключения формы или фрейма, если *aIsSetOpcName* = *True*.

Функция *AddSvgPanel* - подключение SVG панели, ее параметры:

aSvgPanel - указатель на подключаемую SVG панель.

Задание OPC имени динамизируемым объектам

```
procedure TdepOpcApplication.SetOpcNameForComp(aComp: TComponent; const aOpcName: string);
```

Процедура изменяет OPC имя компонента *aComp* на *aOpcName*. При этом все дочерние компоненты также изменят свои OPC имена и соответственно OPC привязки будут работать уже с другими элементами OPC сервера.

```
procedure TdepOpcApplication.SetOpcNameForSvgObjName(aSvgPanel: TdepSvgPanel;  
  const aSvgObjName: string; const aOpcName: string);
```

Процедура изменяет OPC имя SVG объекта с именем *aSvgObjName* на *aOpcName*, принадлежащего SVG панели *aSvgPanel*. При этом все дочерние объекты также изменят свои OPC имена и соответственно OPC привязки будут работать уже с другими элементами OPC сервера.

Блокировка и разблокировка задания OPC имени

```
procedure TdepOpcApplication.LockSendOpcName(const aForm: TCustomForm);
```

Процедура блокирует задание OPC имени для дочерних объектов, лежащих на форме *aForm*, при использовании процедур *SetOpcNameForComp* и *SetOpcNameForSvgObjName*, описанных выше.

```
procedure TdepOpcApplication.UnlockSendOpcName(const aForm: TCustomForm);
```

Процедура изменяет OPC имена для дочерних объектов, лежащих на форме *aForm*, OPC имена, которых заданы при использовании процедур *SetOpcNameForComp* или *SetOpcNameForSvgObjName* и вызванных после процедуры *LockSendOpcName*.

Данные методы рекомендуется использовать для оптимизации скорости работы приложения при многократных вызовах *SetOpcNameForComp* или *SetOpcNameForSvgObjName* подряд. Для этого в начале задания OPC имен вызовите *LockSendOpcName*, в конце *UnlockSendOpcName*.

Синхронное чтение и запись значений в тэги(элементы) OPC сервера

```
property DefaultAlias: TdepOpcAlias read getDefaultAlias;
```

```
property ListAliases: TdepOpcListAliases read fListAliases;
function TdepOpcListAliases.GetAliasOverName(const aName: string): TdepOpcAlias;
```

Запись и чтение значений элементов OPC сервера происходит через методы OPC псевдонимов данных серверов(т.к. *TdepOpcAlias* наследован от *TdepOpcServer*). Для обращения к OPC псевдонимам у класса *TdepOpcApplication* есть два свойства *DefaultAlias* и *ListAliases*. Свойство *DefaultAlias* является указателем на класс *TdepOpcAlias*, который реализует методы и свойства псевдонима по умолчанию. Указатели на все псевдонимы занесены в список *ListAliases*, представляющий собой типизированный список класса *TList*, элементы которого являются указателями на *TdepOpcAlias*. Для получения указателя на *TdepOpcAlias* по его имени можно воспользоваться функцией *GetAliasOverName*

```
function TdepOpcServer.ReadFromTag(const aFullOpcName: string; var aValue: Variant;
var aQuality: TdepOpcQuality; var aTimeStamp: TDateTime): HRESULT;

function TdepOpcServer.WriteToTag(const aFullOpcName: string; const aValue: Variant;
const aQualitySpecified: Boolean = False; const aQuality: TdepOpcQuality = 0): HRESULT;
```

Функция *ReadFromTag* читает значение *aValue*, качества *aQuality* и времени изменения из элемента OPC сервера с его полным именем *aFullOpcName*.

Функция *WriteToTag* записывает значение *aValue*. Качество *aQuality* пишется только в случае, если *aQualitySpecified = True*.

Обе функции возвращают стандартный тип *HRESULT* для контроля ошибок в COM/DCOM.

В *Windows.pas* определены функции *Succeeded* и *Failed*, принимающие в качестве параметра *HRESULT* и возвращающие результат типа *Boolean*. Смысл функций очевиден из их названий: *Succeeded* возвращает *True*, если переданное ей значение кодирует успех, *Failed* - если значение кодирует ошибку. Подключив модуль *ComObj.pas*, можно использовать процедуру *OleCheck*, которая принимает в качестве параметра выражение типа *HRESULT* и создаёт исключительную ситуацию *EoleSysError*, если значение этого выражения кодирует ошибку. В качестве параметра функции *OleCheck* удобно передавать выражение, возвращаемое функциями, имеющими тип *HRESULT*: в случае ошибки возникнет исключение, которое легко обработать в блоке *try/except*.

Асинхронное чтение и запись значений в тэги(элементы) OPC сервера(пользовательские привязки)

```
function TdepOpcServer.CreateItemTag(const aFullOpcName: string = ''; aRequestedType: TVarType = varEmpty;
aOnConnected: TdepOpcItemTagNotify = nil; aOnDisconnected: TdepOpcItemTagNotify = nil;
aOnChange: TdepOpcItemTagChangeNotify = nil; aIfChangedThenRunDataChange: Boolean = True): TdepOpcItemTag;

TdepOpcItemTagNotify = procedure(aOpcItemTag: TdepOpcItemTag) of object;

TdepOpcItemTagChangeNotify = procedure(aOpcItemTag: TdepOpcItemTag;
const aValue: Variant; const aQuality: TdepOpcQuality) of object;
```

Для получения событий об изменении OPC элемента требуется создать объект класса *TdepOpcItemTag*, используя функцию объекта *TdepOpcServer*. Вызывать данную функцию можно также через объект класса *TdepOpcAlias*, т.к. *TdepOpcAlias* наследован от *TdepOpcServer*.

Параметры данной функции:

aFullOpcName - полное имя OPC элемента;

aRequestedType - тип возвращаемого *Value* (возможны значения *varEmpty* - тип значения будет по умолчанию, *varOleStr* - тип значения строка);

aOnConnected - указатель на процедуру, которая будет вызвана при подключении OPC элемента к серверу.

aOnDisconnected - указатель на процедуру, которая будет вызвана при отсоединении OPC элемента от сервера.

aOnChange - указатель на процедуру, которая будет вызвана при изменении *Value* или *Quality*.

aIfChangedThenRunDataChange - указывает вызывать ли *aOnChange*, при создании *TdepOpcItemTag*, если с таким же *aFullOpcName* уже создан *TdepOpcItemTag*.

Пример, создания пользовательской привязки:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, depOpc, depOpcTypes;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Memo1: TMemo;
    procedure Button1Click(Sender: TObject);
  private
    procedure OpcItemConnected(aOpcItemTag: TdepOpcItemTag);
    procedure OpcItemDisconnected(aOpcItemTag: TdepOpcItemTag);
    procedure OpcItemChange(aOpcItemTag: TdepOpcItemTag; const aValue: Variant;
      const aQuality: TdepOpcQuality);
  public
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
  gOpcApplication.DefaultAlias.CreateItemTag('0:Analog:1',
    varEmpty, OpcItemConnected, OpcItemDisconnected, OpcItemChange);
end;

procedure TForm1.OpcItemConnected(aOpcItemTag: TdepOpcItemTag);
begin
  Memo1.Lines.Add('Connected: ' + aOpcItemTag.OpcName);
end;

procedure TForm1.OpcItemDisconnected(aOpcItemTag: TdepOpcItemTag);
begin
  if Memo1 <> nil then
    Memo1.Lines.Add('Disconnected: ' + aOpcItemTag.OpcName);
end;

procedure TForm1.OpcItemChange(aOpcItemTag: TdepOpcItemTag; const aValue: Variant;
  const aQuality: TdepOpcQuality);
begin
  Memo1.Lines.Add('Changed: Value = ' + VarToStr(aOpcItemTag.Value) +
    ' Quality = ' + IntToStr(aQuality));
end;

end.
```

Тип TdepOpcQuality и функции для работы с ним

```
TdepOpcQuality = WORD;
```

Тип TdepOpcQuality является переопределенным типом WORD, что позволяет хранить в нем данные в соответствии со

спецификацией OPC Data Access.

```
procedure QualitySetAlarmed(var aQuality: TdepOpcQuality; const b: Boolean);
```

Процедура устанавливает или сбрасывает бит внимания(тревоги) в соответствии с параметром *b* (*True* - установка, *False* - сброс).

```
function QualityGetAlarmed(const aQuality: TdepOpcQuality): Boolean;
```

Функция получает значение бита внимания(тревоги).

```
procedure QualitySetUser(var aQuality: TdepOpcQuality; const b: Boolean);
```

Процедура устанавливает или сбрасывает пользовательский бит в соответствии с параметром *b* (*True* - установка, *False* - сброс).

```
function QualityGetUser(const aQuality: TdepOpcQuality): Boolean;
```

Функция получает значение пользовательского бита.

```
procedure QualitySetLimitLow(var aQuality: TdepOpcQuality; const b: Boolean);
```

Процедура устанавливает или сбрасывает бит нижнего предела в соответствии с параметром *b* (*True* - установка, *False* - сброс).

```
function QualityGetLimitLow(const aQuality: TdepOpcQuality): Boolean;
```

Функция получает значение бита нижнего предела.

```
procedure QualitySetLimitHigh(var aQuality: TdepOpcQuality; const b: Boolean);
```

Процедура устанавливает или сбрасывает бит верхнего предела в соответствии с параметром *b* (*True* - установка, *False* - сброс).

```
function QualityGetLimitHigh(const aQuality: TdepOpcQuality): Boolean;
```

Функция получает значение бита верхнего предела.

```
procedure QualitySetGood(var aQuality: TdepOpcQuality);
```

Процедура устанавливает "хорошее" значение качества.

```
function QualityIsGood(const aQuality: TdepOpcQuality): Boolean;
```

Функция возвращает *True*, если качество "хорошее", в противном случае - *False*.

```
procedure QualitySetBad(var aQuality: TdepOpcQuality);
```

Процедура устанавливает "плохое" значение качества.

```
function QualityIsBad(const aQuality: TdepOpcQuality): Boolean;
```

Функция возвращает *True*, если качество "плохое", в противном случае - *False*.

```
function QualityIsLastKnownValue(const aQuality: TdepOpcQuality): Boolean;
```

Функция возвращает *True*, если в *Value* содержится последнее достоверное значение.

9.5.5 Дополнительные компоненты

9.5.5.1 Поддержка SVG графики

9.5.5.1.1 Формат SVG-объекта

SVG (от англ. *Scalable Vector Graphics* — масштабируемая векторная графика; произносится [эс-ви-джи]) — язык разметки масштабируемой векторной графики, созданный Консорциумом Всемирной паутины (W3C) и входящий в подмножество расширяемого языка разметки XML, предназначен для описания двумерной векторной и смешанной векторно/растровой графики в формате XML. Поддерживает как неподвижную, так анимированную и интерактивную графику — или, в иных терминах, декларативную и скриптовую. Это открытый стандарт, является рекомендацией консорциума W3C, — организации, разработавшей такие стандарты, как HTML и XHTML. В основу SVG легли языки разметки VML и PGML.

Достоинства формата

- Текстовый формат — файлы SVG можно читать и редактировать при помощи обычных текстовых редакторов. При просмотре документов, содержащих SVG графику, имеется доступ к просмотру кода просматриваемого файла и возможность сохранения всего документа. Кроме того, SVG файлы обычно получаются меньше по размеру, чем сравнимые по качеству изображения в форматах JPEG или GIF, а также хорошо поддаются сжатию.
- Масштабируемость — SVG является векторным форматом. Существует возможность увеличить любую часть изображения SVG без потери качества. Дополнительно, к элементам SVG документа возможно применять фильтры — специальные модификаторы для создания эффектов, подобным применяемым при обработке растровых изображений (размытие, выдавливание, сложные системы трансформации и др.) В тексте SVG-кода фильтры описываются тегами, визуализацию которых обеспечивает средство просмотра, что не влияет на размер исходного файла, обеспечивая при этом необходимую иллюстративную выразительность.
- Широко доступно использование растровой графики в SVG документах. Имеется возможность вставлять элементы с изображениями в форматах PNG, GIF или JPG.
- Текст в графике SVG является текстом, а не изображением, поэтому его можно выделять и копировать, он индексируется поисковыми машинами, не нужно создавать дополнительные метафайлы для поисковых серверов.
- Анимация реализована в SVG с помощью языка SMIL (Synchronized Multimedia Integration Language), разработанного также консорциумом W3C. Поддерживаются скриптовые языки на основе спецификации ECMAScript. SVG-элементами можно управлять с помощью JavaScript. Применение скриптов и анимации в SVG позволяет создавать динамичную и интерактивную графику. В SVG обеспечивается событийная модель, отслеживаются события (загрузка страницы, изменение ее параметров, события мыши, клавиатуры и др.) Анимация может запускаться по определенному событию (например «onmouseover» или «onclick»), что придаёт графике интерактивность. У каждого элемента есть свои собственные события, к которым можно привязывать отдельные скрипты.
- SVG — открытый стандарт. В отличие от некоторых других форматов, SVG не является чьей-либо собственностью.
- SVG документы легко интегрируются с HTML и XHTML документами. Внешний SVG подключаются через тег `<embed>`, значение атрибута `src` имя файла с расширением «.svg», содержащего разметку SVG. Атрибуты `width` и `height` определяют размеры области SVG по-горизонтали и по-вертикали. Элементы SVG совместимы с HTML и DHTML.
- Совместимость с CSS (англ. Cascading Style Sheets). Отображением (форматированием и декорированием) SVG элементов можно управлять с помощью таблицы стилей CSS 2.0 и её расширений, либо напрямую с помощью атрибутов SVG элементов.
- SVG предоставляет все преимущества XML:
- Возможность работы в различных средах.
- Интернационализация (поддержка Юникода).
- Широкая доступность для различных приложений.
- Лёгкая модификация через стандартные API — например, DOM. SVG поддерживает стандартизованную W3C объектную модель документа DOM, обеспечивая доступ к любому элементу, что даёт широкие возможности по динамичному

модифицированию элементов их атрибутов и событий.

- Лёгкое преобразование таблицами стилей XSLT. Как любой основанный на XML формат, SVG дает возможность использовать для его обработки таблицы трансформации (XSLT). Преобразуя XML-данные в SVG с помощью простого XSL, можно легко получить графическое представление любых данных, например визуализировать химические молекулы, описанных на языке CML (Chemical Markup Language).

Подробнее на <http://ru.wikipedia.org/wiki/SVG>.

Для создания SVG файлов рекомендуется использовать открытый графический редактор Inkscape (<http://ru.wikipedia.org/wiki/Inkscape>).

Отображение и взаимодействие с SVG объектами осуществляется через компонент TdepSvgPanel. Компонент TdepSvgPanel работает с SVG-клонами как с отдельными объектами. OPC привязки для клона наследуются из оригинала, при динамизации в клоне можно только поменять OPC имя SVG-объекта и добавить новые привязки. Тем самым клоны можно рассматривать как подбине VCL-фреймов.

```
TdepSvgPanel = class(TWinControl)
public
    constructor Create(aOwner: TComponent); override;
    destructor Destroy; override;
    procedure Reload(); virtual;
    procedure ChangeFileNameDialog();
    procedure SetFilePath(const aFilePath: string);
    procedure LockPaint();
    procedure UnlockPaint();
    property Design: Boolean read getDesign write setDesign;
    property FullFileName: string read getFullFileName;
    property CurSvgHint: string read fCurSvgHint write setCurSvgHint;
published
    property FileName: string read fFileName write setFileName;
    property Scale: Single read getScale write setScale;
    property Proportional: Boolean read getProportional write setProportional default True;
    property Stretch: Boolean read getStretch write setStretch default True;
    property Center: Boolean read getCenter write setCenter default True;
    property BackgroundColor: TColor read getBackgroundColor write setBackgroundColor;
    property AutoReloadIfModified: Boolean read fAutoReloadIfModified write setAutoReloadIfModified default True;
    property QueryReloadIfModified: Boolean read fQueryReloadIfModified write setQueryReloadIfModified default True;
    property OnClickSvgObjs: TdepSvgOnEventSvgObjs read fOnClickSvgObjs write setOnClickSvgObjs;
    property OnMouseMoveSvgObjs: TdepSvgOnEventSvgObjs read fOnMouseMoveSvgObjs write setOnMouseMoveSvgObjs;
    property Cursor: TCursor read getCursor write setCursor;
    property TypeRendering: TdepSvgTypeRendering read fTypeRendering write setTypeRendering default trNat;
    property SvgHint: string read fSvgHint write setSvgHint;
    property ShowSvgHint: Boolean read fShowSvgHint write setShowSvgHint default True;
end;
```

9.5.5.1.2 Новые возможности Inkscape

Бывают случаи, когда необходимо построить АРМ, состоящий из однотипных схем, а каждая схема состоит из однотипных элементов.

Построение схемы, состоящей из однотипных элементов, решается применением клонов и образцов. Но если надо создать набор файлов-схем, то тут начинаются проблемы. Проблемы связаны с тем, что нельзя использовать один и тот же оригинал в нескольких файлах и создавать от него клоны в разных файлах.

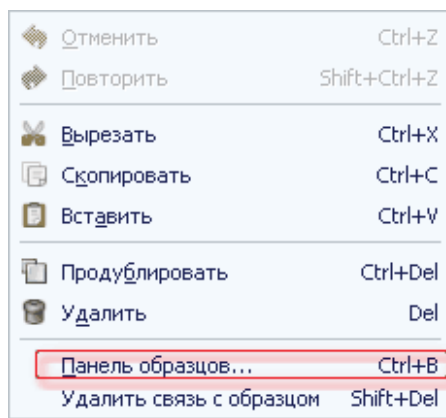
Для решения этой проблемы мы модифицировали Inkscape. Для установки новой версии необходимо заменить папку C:\Program Files\Inkscape на новую папку Inkscape, поставляемую с диском.

Расширение нового Inkscape заключается в применении библиотеки образцов.

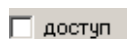
Библиотека образцов представляет собой набор графических svg - элементов, расположенных в файлах папки с именем

patterns. Папка patterns лежит в том месте, где сохранен текущий файл. Для того, чтобы svg - элемент был образцом необходимо, чтобы он являлся группой.

После того, как вы сохранили текущий файл, с помощью вызова правой кнопки мыши "Панель образцов":



появится панель "Панель образцов", которая отображает все элементы, находящиеся в папке patterns:



Элементы, находящиеся на этой панели, можно тащить на канву. Обычно элемент кладут за пределы видимости документа, а созданные от него клоны делают видимыми.

Динамизацию необходимо начать с файлов образцов. Все файлы, использующие образцы, автоматически подхватят привязки. В файлах можно добавлять дополнительные привязки и задавать имена OPC-компонентов.

9.5.5.2 Работа с пользователями

Для работы с пользователями в [Настройках OPC проекта](#) должна быть включена поддержка OPC пользователей и подключен заголовочный файл `depOpc.hpp`. После включения поддержки OPC пользователей у глобального объекта `gOpcApplication` будет доступно свойство `Admin` типа `TdepOpcAdmin`. Вся дальнейшая работа с пользователями производится через это свойство.

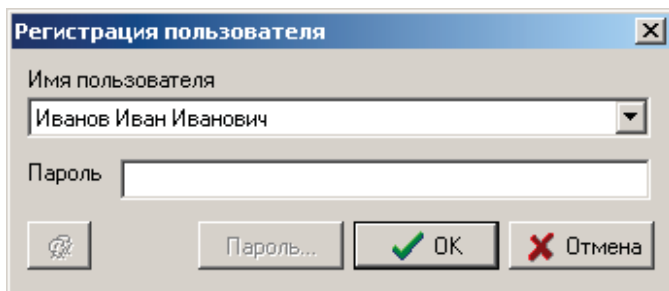
Перед использованием компонента `TdepOpcAdmin` требуется указать с каким OPC сервером(где находятся пользователи) будет производиться работа. Например,

```
gOpcApplication->Admin->NameOpcAlias = gOpcApplication->DefaultAlias->Name;
```

или

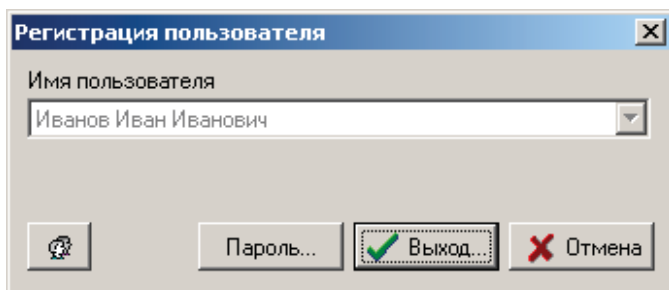
```
gOpcApplication->Admin->OpcAlias = gOpcApplication->DefaultAlias;
```

Чтобы обеспечить регистрацию OPC-клиента в модели, нужно воспользоваться методом `LogonDlg`. Далее нужно задать `OpcName` - путь к элементу `iUsers` модели. Потом, например в обработчике нажатия на некоторую кнопку, нужно вызвать метод `TdepAdmin.LogonDlg()`. При первом вызове этого метода появится диалог регистрации пользователя



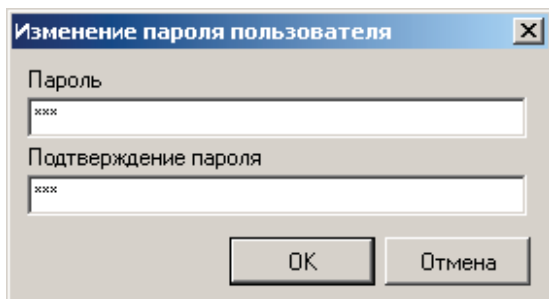
в котором нужно выбрать нужное имя пользователя, задать пароль и нажать кнопку ОК. Если диалог закроется без сообщений, то значит регистрация произведена успешно.

Если вызвать метод TdepOpcAdmin после регистрации пользователя, то появится следующий диалог:




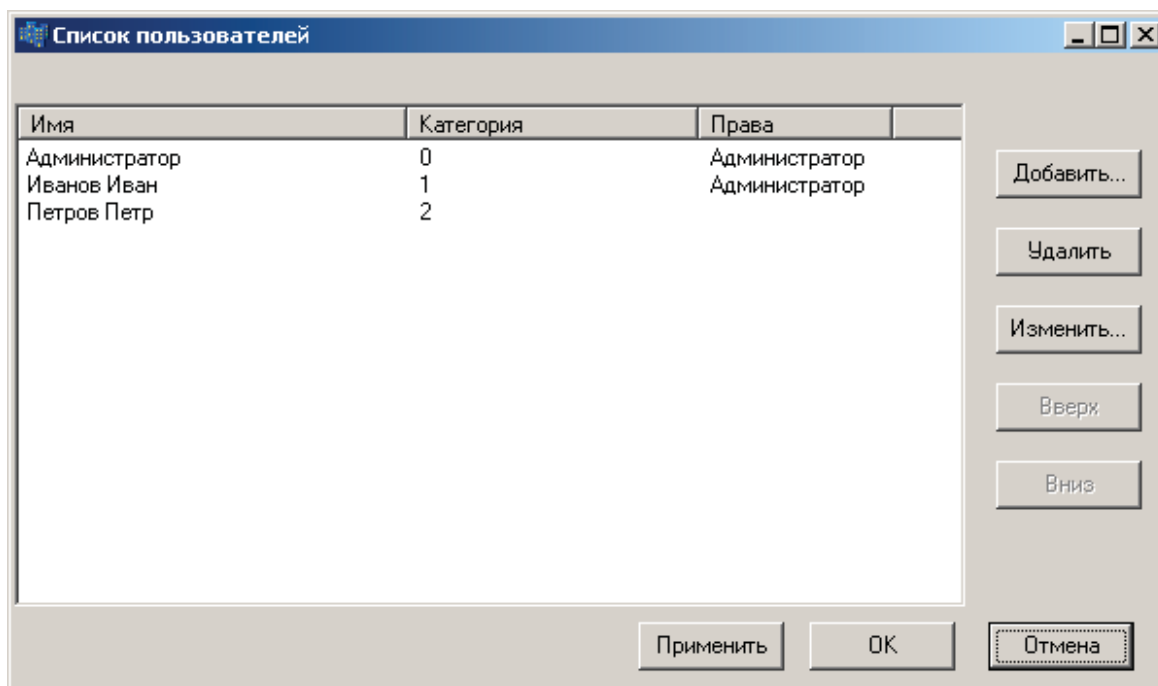
Для разрегистрации пользователя нужно нажать кнопку "Выход...".

Для изменения пароля нужно нажать кнопку "Пароль", после чего появится следующий диалог



в котором нужно задать новый пароль.

Если зарегистрированный пользователь имеет права Администратора, то в диалоге "Регистрация пользователя" ему доступна кнопка "Пользователи"  при нажатии на которой, появится следующий диалог:



в котором Администратор может добавлять и удалять пользователей и менять их параметры.

Приведем неполное объявление класса TdepOpcAdmin:

```
class TdepOpcAdmin : public TObject
{
...
public:
__property AnsiString OpcName; // Путь к элементу iUsers модели.
__property TNotifyEvent OnLogon; // Событие, срабатывающее после регистрации или разрегистрации текущего
TdepOpcUser* CurUser; // текущий пользователь
TDateTime LogonTime; // Время регистрации текущего пользователя

__property bool Logged; // пользователь зарегистрирован

void LogonDlg(); // Диалог регистрации
void LogoffDlg(); // Диалог разрегистрации
void UsersDlg(); // Диалог редактирования списка пользователей
void PwdChDlg(); // Диалог изменения пароля
bool PwdEnterDlg(); // Вызов диалога проверки пароля

TdepOpcUser* UserByName(const AnsiString& n); // Поиск пользователя по имени.
};
```

Текущий пользователь CurUser - это пользователь, который зарегистрирован в данный момент или был последним зарегистрированным пользователем. TdepOpcAdmin содержит в себе список доступных пользователей TdepOpcUser.

```
class TdepOpcUser : public TObject
{
public:
__property bool Valid; // Можно ли обращаться к свойствам данного элемента
```

```

__property AnsiString UserName; // Имя пользователя
__property bool IsAdmin; // Признак Администратора
__property bool IsUser; // Признак Пользователя
__property AnsiString CtgCaption; // Название категории
__property int CtgID; // Номер категории
__property AnsiString Pwd; // Пароль
bool Logged; // Зарегистрирован

void Logon(const AnsiString& APwd); // Зарегистрировать
void Logoff(); // Разрегистрировать
};

```

Чтобы обработать ситуацию регистрации и раз регистрации пользователя используют событие OnLogon, в котором по CurUser определяется текущий пользователь и по Logged определяется зарегистрировался ли он или раз зарегистрировался.

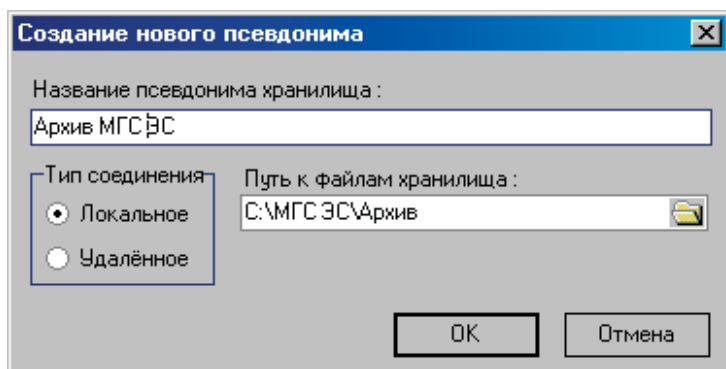
Диалоги LogoffDlg, UsersDlg и PwdChDlg вызываются при нажатии на соответствующие кнопки диалога LogonDlg.

9.5.5.3 Работа с оперативным журналом

Оперативный журнал позволяет архивировать последовательность событий и квитиовать их. Запись журнала содержит следующие поля:

int	ID	уникальный в течение суток идентификатор не равный 0
TDateTime	DateTime	зимнее время возникновения события
String	Action	действие, характеризующее событие, например "Работа", "Останов", "Открытие" и тд.
String	Object	имя элемента, связанного с событием, например, "КП1\Насос2"
String	Disp	имя диспетчера, зарегистрированного в момент возникновения события
WORD	Severity	Severity - показывающее важность события от 1(Уведомление) до 1000(Авария)
String	OpcName	полное OPC имя элемента Object, например, "CP1\Pump2"
String	Comment	комментарий
bool	Kvit	квитиовано ли событие
TDateTime	KvitDateTime	зимнее время квитиования
String	KvitDisp	Диспетчер, квитиовавший событие.

Для работы с оперативным журналом библиотека содержит визуальный компонент TdepArcOperLog.



Компонент представляет собой таблицу, колонки которой соответствуют полям оперативного журнала.

Номер	Дата	Важность	Название важности	Событие	Объект	Диспетчер	Квитировано	Квит. Дисп.	Время квит.
1	24.04.2006 18:00:42	🚨	Тревога	Авария	КП1\Насос2		✓	Иванов Иван Иванович	24.04.2006 19:00:46
2	24.04.2006 18:00:40	🚨	Тревога	Останов	КП1\Насос2		✓	Иванов Иван Иванович	24.04.2006 19:00:48
3	24.04.2006 18:00:35	🚨	Тревога	Авария	КП1\Насос1				0:00:00
4	24.04.2006 18:00:32	🚨	Тревога	Останов	КП1\Насос1				0:00:00
5	24.04.2006 18:00:19	👤	Параметры пользователя	Регистрация пользователя	Пользователи Иван Иван	Иванов Иван Иванович	✓	Иванов Иван Иванович	24.04.2006 19:00:22
6	24.04.2006 17:59:34	👤	Параметры пользователя	Регистрация пользователя	Пользователи Иван Иван	Иванов Иван Иванович	✓	Иванов Иван Иванович	24.04.2006 19:00:24
7	24.04.2006 17:59:32	👤	Параметры пользователя	Выход пользователя	Пользователи Администратор	Администратор	✓	Иванов Иван Иванович	24.04.2006 19:00:25
8	24.04.2006 17:59:26	👤	Параметры	Регистрация	Пользователи	Администратор			0:00:00

Для использования компонента ему необходимо указать хранилище (свойство TdepArcOperLog.Storage).

Колонка "Название важности" рассчитывается по таблице соответствия номера важности и его названия. Эта таблица содержится вместе с оперативным журналом.

Компоненты столбцов реализованы как под-компоненты TdepArcOperLog.

```
// Тип столбца.
class TdepArcOperLogColumn : public TComponent
{
    __property AnsiString Caption; // Заголовок. Если нужно, чтобы заголовок был написан в две строчки, то
    __property int Width; // Ширина. Устанавливается при проектировании компонента.
    __property bool Visible; // Показывать эту колонку
    __property int OrderIndex; // Порядковый номер колонки (начинается с нуля)
public:
    __fastcall TdepArcOperLogColumn(TComponent* AOwner);
    __fastcall TdepArcOperLogColumn(TComponent* AOwner, const AnsiString& ACapt, int AWidth, int AIndex=-1);
    __fastcall ~TdepArcOperLogColumn();
};
```

Некоторые столбцы имеют тип, производный от TdepArcOperLogColumn, например TdepArcOperLogDateTime-дата возникновения события и TdepArcOperLogSeverity-категория.

```
enum TdepDTSortType {dtsTimeUp, dtsTimeDown}; // Тип сортировки записей в таблице. При dtsTimeUp более поздние записи
показываются выше.
```


// Дата возникновения события.

```
class TdepArcOperLogDateTime : public TdepArcOperLogColumn
{
    __published :
        __property TdepArcOperLogDTSortType SortType; // Тип сортировки записей в таблице.
public :
    __fastcall TdepArcOperLogDateTime(TComponent* AOwner);
    __fastcall TdepArcOperLogDateTime(TComponent* AOwner, const AnsiString& ACapt, int AWidth, int AIndex=-1);
    __fastcall ~TdepArcOperLogDateTime();
};
```

// Важность события

В зависимости от важности события записи в таблице можно изобразить по-разному.

Важности объединяются в интервалы значений, каждый из которых имеет свои параметры отображения.

```
class TdepArcOperLogSeverity : public TdepOLColumn
{
    __published :
        __property TOwnedCollection *Intervals; // Список параметров интервалов.
        __property TImageList *Images; // Картинки, используемые для показа в столбце "Категория".
public :
    __fastcall TdepArcOperLogSeverity(TComponent* AOwner);
    __fastcall TdepArcOperLogSeverity(TComponent* AOwner, const AnsiString& ACapt, int AWidth, int AIndex=-1);
    __fastcall ~TdepArcOperLogSeverity();

    TdepOLSevIvl* SevIvlByCat(int ACategory);
};
```

// Параметры интервала

```
class TdepArcOperLogSevIvl : public TCollectionItem
{
    __published :
        __property WORD LowVal; // Нижняя граница интервала
        __property WORD HighVal; // Верхняя граница интервала
        __property int ImageIndex; // Индекс картинки из TdepOLCategory.Images
        __property TFont *Font; // Шрифт текста в записи события
        __property TColor Color; // Цвет фона в записи события
public :
    __fastcall TdepArcOperLogSevIvl(TCollection* Collection);
    __fastcall ~TdepArcOperLogSevIvl();
};
```

Если категория события не входит ни в один интервал, то событие отображается с цветом фона TdepArcOperLog.Color и шрифтом TdepArcOperLog.Font.

// Тип показа оперативного журнала.

```
enum TdepOLType {oltTime, oltCount};
```

Если oltTime, то показывается содержимое журнала то TimeBegin до TimeEnd.

Если oltCount, то показываются последние EventCount записей.

// Оперативный журнал.

```
class TdepArcOperLog : public TCustomDrawGrid
{
public:
```

```
__property TdepOpcAlias *OpcAlias; // Указатель на Opc псевдоним. Используется при квитировании записи.

__published :
__property TdepArcStorage *Storage; // Хранилище, содержащее оперативный журнал
__property bool KvitNeeded; // Возможно ли квитиование
__property bool KvitAck; // Спрашивать ли подтверждение при квитировании
__property DWORD EventCount; // Количество записей, если OlType==oltCount
__property TdepArcOperLogType OlType; // Тип показа
__property AnsiString OpcName; // OPC-Имя элемента модели типа iOperLog.
__property TdepArcOperLogDateTime *ColDateTime; // Дата возникновения события
__property TdepArcOperLogSeverity *ColSeverity; // Категория
__property TdepArcOperLogColumn *ColAction; // Действие
__property TdepArcOperLogColumn *ColObject; // Название элемента, связанного с событием.
__property TdepArcOperLogColumn *ColDisp; // Диспетчер, во время работы которого произошло событие.
__property TdepArcOperLogColumn *ColKvit; // Квитиовано ли событие
__property TdepArcOperLogColumn *ColKvitDisp; // Диспетчер, квитиовавший событие
__property TdepArcOperLogColumn *ColKvitDT; // Дата квитации.
__property TdepArcOperLogColumn *ColNo; // Порядковый номер записи в таблице, начиная с 1.
__property bool KvitShow; // Показывать-ли квитиованные события.
__property int CaptHeight; // Высота строки заголовков.
__property TColor CaptColor; // Цвет строки заголовков.
__property TColor GridLineColor; // Цвет линий сетки
__property AnsiString BoundsStr; // Строка, указывающая допустимые для показа важности событий журнала.
__property int ValidDays; // "Актуальность выводимых данных", 0-все, N- только за N последних дней. Если
__property bool AutoSizeCols; // При установленном свойстве, если размеры окна оперативного журнала меня
// в которые выводятся строки, меняют свою ширину пропорционально изменению ширины окна опера

// Свойства унаследованные от базовых классов.
__property Align;
__property Anchors;
__property BiDiMode ;
__property BorderStyle;
__property Color;
__property Constraints ;
__property Ctl3D;
__property DefaultRowHeight = {default=24};
__property Enabled;
__property RowCount = {default=2};
__property Font;
__property GridLineWidth; // Ширина линий сетки
__property ParentBiDiMode;
__property ParentColor;
__property ParentCtl3D;
__property ParentFont;
__property ParentShowHint;
__property PopupMenu;
__property ScrollBars;
__property ShowHint;
__property TabOrder;
__property Visible;
__property VisibleColCount;
__property VisibleRowCount;
public:
__property TdepArcOperLogColumn *Columns[int Index]; // Столбец по его индексу
__property int ColumnCount; // Количество столбцов
__property TDateTime TimeBegin; // Время начала и
__property TDateTime TimeEnd; // конца интервала просмотра если OlType==oltTime
```

```
TdepArcOperLogColumn* ColumnByCol(int ACol); // Столбец по его видимому индексу.
void SetTimeInterval(TDateTime Begin, TDateTime End); // Установить интервал просмотра
};
```

Строка BoundsStr имеет формат N1-N11;N2-N22;...

где N1, N2 ... - нижние границы допустимых интервалов, а N11, N22, ... - верхние границы. Если событие журнала имеет важность, не входящий ни в один интервал, то такое сообщение не показывается. Если строка BoundsStr пустая, то показываются все сообщения.

Для использования компонента нужно задать свойство Storage и соединиться с хранилищем. При этом компонент может только показывать сообщения журнала. Если нужно квити́ровать сообщения, то нужно установить свойство OPCDesigner, установить свойство KvitNeeded в true и установить свойство OpсName указывающим на оперативный журнал в модели.

В свойстве ColSeverity.Intervals можно задать диапазоны событий, каждый из которых имеет свой цвет, картинку и шрифт.

9.5.5.4 Голосовое сопровождение

Компонент "Голосовое сопровождение" TdepSpeak автовыбор предназначен для проговаривания событий оперативного журнала.

Компонент имеет следующие свойства:

```
__property TdepArcStorage *Storage // хранилище, содержащее оперативный журнал
__property TdepSPField *FldSeverity // поле "Важность" журнала
__property TdepSPField *FldAction // поле "Действие" журнала
__property TdepSPField *FldObject // поле "Объект" журнала
__property AnsiString BoundsStr // строка, указывающая допустимые диапазоны важности событий
__property bool KvitdSpeak // проговаривать ли квити́рованные события
__property TSpeakGender Gender // женский или мужской голос
__property DWORD Speed // скорость
__property WORD Pitch // тон
__property WORD Volume // громкость
__property String DicFileName // файл словаря
__property int MaxBufSize // максимальный размер буфера
```

```
__property String      MaxBufStr // строка, проговариваемая при превышении максимального размера буфера.  
__property bool       Active     // признак активности компонента.  
__property bool       ActiveOperLog // признак активности по отношению к оперативному журналу.
```

и методы:

```
void Speek(const AnsiString& AStr); // проговаривание текста AStr  
void GeneralDlg(); // Вызов диалога настроек "движка"
```

Тип TdepSPField обозначает поле оперативного журнала и имеет следующие поля:

```
__property bool      Enabled // проговаривать данное поле  
__property int       OrderIndex // индекс в списке полей.
```

Для использования компонента нужно задать свойство Storage и соединиться с хранилищем, содержащим оперативный журнал.

По умолчанию проговариваются поля FldAction и FldObject, то есть их свойства Enabled установлены в true.

Строка BoundsStr имеет то же значение, что и для компонента "Оперативный журнал" TdepOperLog.

Если свойство MaxBufSize больше нуля, то при проговаривании происходит проверка размера буфера, содержащего проговариваемый текст.

Если буфер превышает размер MaxBufSize, то проговаривание останавливается, текст теряется и проговаривается строка MaxBufStr. Далее компонент работает в обычном режиме.

Если свойство Active стоит в false, то компонент ничего не проговаривает.

Если Active true а ActiveOperLog false, то компонента не проговаривает события оперативного журнала, но проговаривает строки, при программном вызове метода Speek.

Если Active true и ActiveOperLog true, то проговариваются и события оперативного журнала и строки при программном вызове метода Speek.

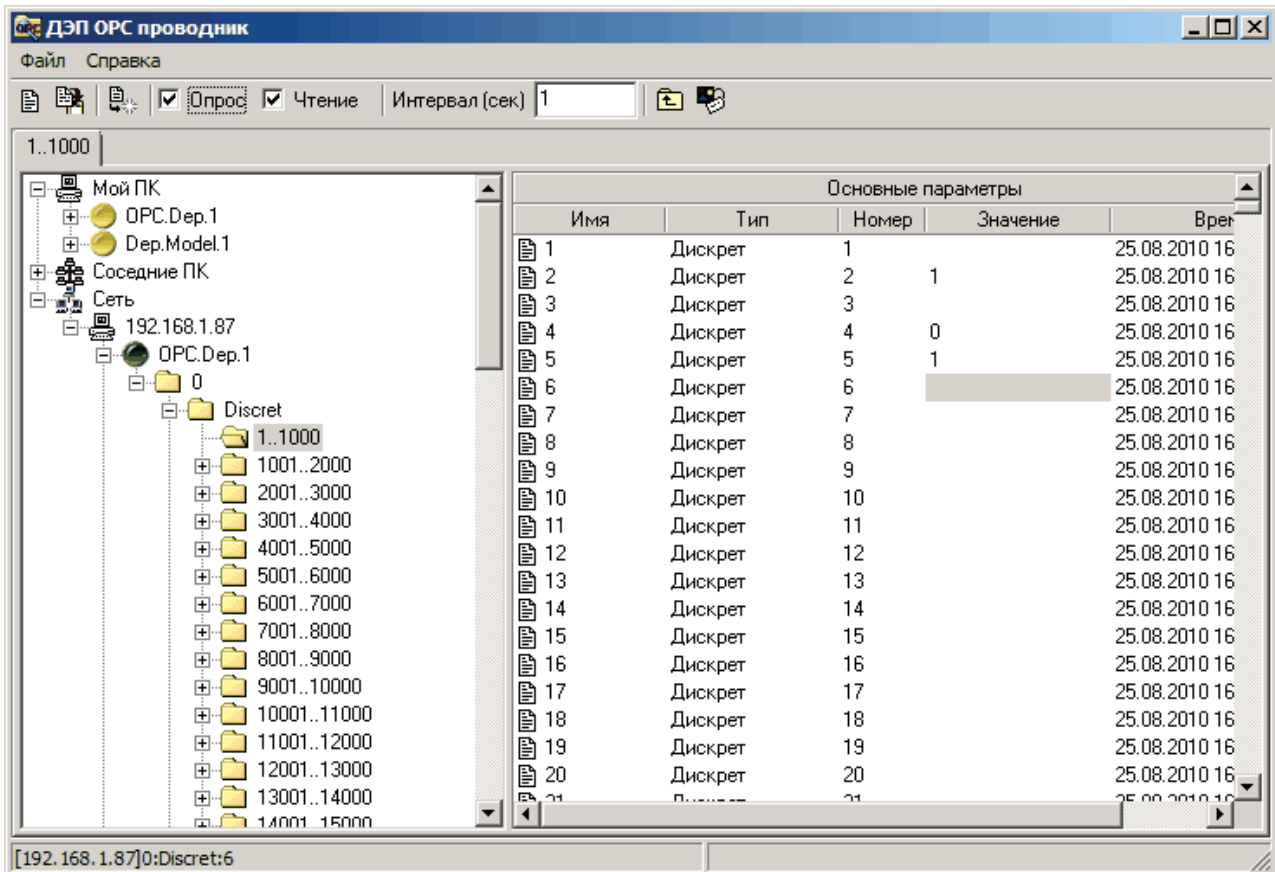
9.6 ДЭП OPC Проводник

9.6.1 Введение

Программа "ДЭП OPC проводник" предназначена для отображения элементов OPC-сервера и изменения их значений. В настоящее время возможна работа только с серверами "OPC Dep" и "Dep Model" поддерживающих стандарт OPC Data Access 3.0.

9.6.2 Интерфейс программы

Главное окно программы содержит вкладки. Вкладки можно добавлять, удалять и делать копию. В настоящее время существуют только вкладки типа "Вид" в которых можно просмотреть содержимое OPC-сервера. Вкладка "Вид" разделена на две части: слева - дерево доступных OPC серверов и их элементов, справа - список подэлементов выбранного слева элемента и их значения.



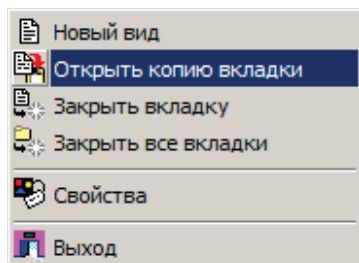
В программе заведен таймер опроса серверов и чтения данных. Включение таймера и его период управляется на панели. Период задается в секундах как число с плавающей точкой.



Заголовок вкладки определяется по имени текущего выбранного в дереве узла.

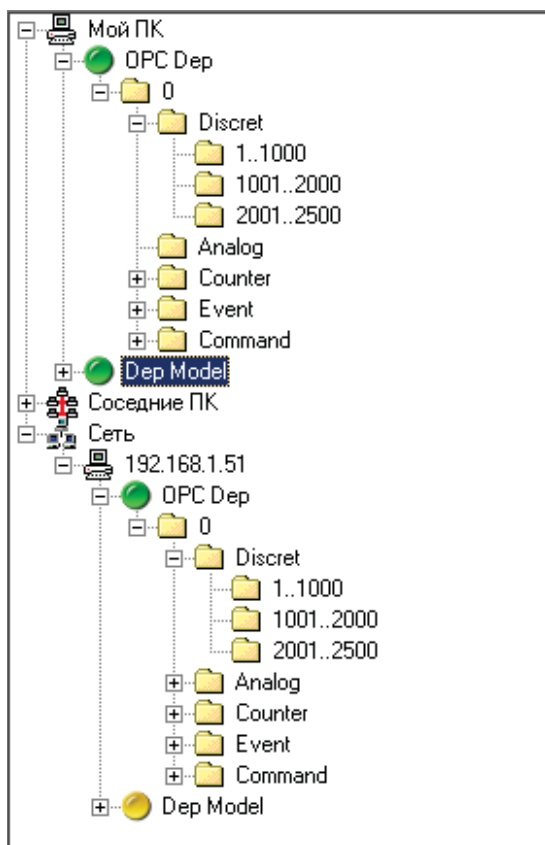
В строке состояния показан OPC ItemID выбранного в списке элемента.

9.6.2.1 Меню файл



- "Новый вид" - создается новая вкладка типа "Вид".
- "Открыть копию вкладки" - создается копия текущей активной вкладки.
- "Заккрыть вкладку" - активная вкладка закрывается.
- "Заккрыть все вкладки" - закрываются все вкладки.
- "Свойства" - открывается диалог "Свойства".
- "Выход" - выход из программы.




9.6.2.2 Дерево



Типы элементов дерева:

- "Мой ПК" - содержимое компьютера, на котором работает программа.
- "Соседние ПК" - содержит список компьютеров из той же рабочей группы, что и данный.
- "Сеть" - содержит список дополнительных узлов сети. Этот список управляется пользователем. Пользователь может добавить узел, удалить узел, изменить имя узла. Список узлов сохраняется в реестр.

Каждый из компьютеров в узлах дерева содержит пару потенциально доступных OPC-серверов: "OPC Dep" и "Dep Model".

OPC-сервер может быть в одном из трех состояний и обозначается следующими иконками:  - начальное состояние,  - с сервером установлена связь,  - состояние ожидания. После установления соединения (по команде пользователя) с сервером сервер переходит из начального состояния в состояние соединения. Далее, если по команде пользователя соединение с сервером разрывается, то сервер переходит обратно в начальное состояние. Если связь с сервером разрывается не по инициативе пользователя (например, выгружается OPC-сервер), то сервер переходит в состояние ожидания. При этом с некоторым тактом программа пытается установить соединение с данным сервером.

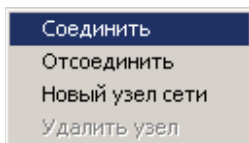
Чтобы вывести сервер из состояния ожидания в начальное нужно дать ему команду "Отсоединить" из всплывающего меню.

При рассоединении с сервером, иконки его элементов перечеркиваются: .

При восстановлении соединения с сервером (при переходе из состояния ожидания в соединенное состояние) происходит проверка "старого" и "нового" дерева элементов так, чтобы дерево по возможности сохранилось больше, то есть, если на некотором уровне обнаружилось различия, то этот уровень перестраивается.

Если OPC элемент дерева имеет больше 1000 подэлементов, то для удобства работы, все подэлементы разбиваются на группы по 1000 штук и каждая группа помещается в свой узел. В заголовке такого узла показано: с какого- по какой элемент находятся в данном узле.

Дерево имеет всплывающее меню:



- Соединить - установить соединение с выбранным в дереве сервером.
- Отсоединить - разорвать соединение с выбранным сервером.
- Новый узел сети - добавить новый узел в список "Сеть".
- Удалить узел - удалить узел из списка "Сеть".

9.6.2.3 Список

Имя	Название	Вни...	Значение
Caption		<input type="checkbox"/>	1
Pitanie	Питание	<input type="checkbox"/>	
tRele	Тепловое реле	<input type="checkbox"/>	
Opened	Открыта	<input type="checkbox"/>	
Closed	Закрота	<input type="checkbox"/>	
CtrlMode	Режим управления	<input type="checkbox"/>	
State	Состояние	<input type="checkbox"/>	
Avaria	Авария	<input type="checkbox"/>	
Info	Информация	<input type="checkbox"/>	
ModeAuto	Режим АВТО	<input type="checkbox"/>	
ModeLocal	Режим МЕСТНЫЙ	<input type="checkbox"/>	
ModeDisp	Режим ДИСПЕТЧЕР	<input type="checkbox"/>	
DispOpen	Команда ДИСП ОТКРЫТЬ	<input type="checkbox"/>	
DispStop	Команда ДИСП СТОП	<input type="checkbox"/>	
DispClose	Команда ДИСП ЗАКРЫТЬ	<input type="checkbox"/>	
Position	Положение задвижки	<input type="checkbox"/>	23.3455
Alarm	Аварийное событие	<input type="checkbox"/>	0
Warning	Предупреждение	<input type="checkbox"/>	0

Список подэлементов выбранного в дереве элемента, располагается в правой части вкладки "Вид". Для "OPC Dep" сервера список имеет столбцы: "Имя", "Тип", "Номер", "Значение" и "Время". Возможные значения поля тип: Дискрет, Аналог, Счетчик, Событие, Команда, Аналог-событие, Аналог-команда, Счетчик-событие, Счетчик-команда. Поле "Номер" означает номер элемента в базе контроллера Windecop. Для "Dep Model" список имеет следующие столбцы: "Имя", "Название", "Внимание", "Значение" и "Время". Элементы, содержащие внутри себя подэлементы отображаются со значком папки. При двойном щелчке мышью или при нажатии клавиши "ввод" на таком элементе, мы проваливаемся на уровень вниз как в обычном проводнике Windows.

С тактом, указанным в периоде опроса, происходит чтение значения и качества видимых элементов списка.

Если значение не определено, то поле "Значение" пустое.

Если значение не определено, но известно последнее определенное значение, то это значение показывается светло-серым цветом.

Если значение определено, то оно показывается черным цветом.

Если значение получено не удалось (нет соединения с сервером или элемент не содержит OPC-значение) то в поле "Значение" показывается "???".

Из окна списка можно записать значение в OPC элемент изменив поле "Значение".

Строковые значения выделяются кавычками.

9.6.2.4 Запись

Имя	Текущее		Новое	
	Внима...	Значение	Внима...	Значение
Caption	<input type="checkbox"/>	23	<input type="checkbox"/>	

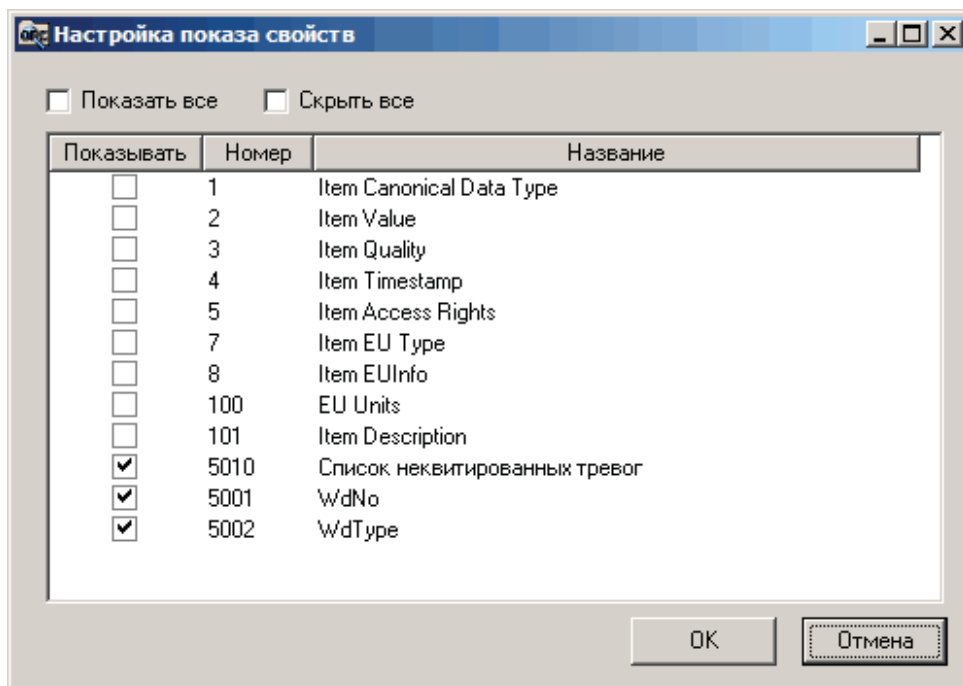
OK Отмена Применить

Диалог записи значений вызывается из всплывающего меню списка значений.

Пока работает запись только одного элемента.

Для записи значения нужно изменить поля "Внимание" и "Значение" в разделе "Новое" и нажать "Применить".

9.6.2.5 Свойства



Диалог "Свойства" позволяет настраивать показ свойств в списке элементов сервера. Список свойств пополняется по мере опроса новых элементов сервера.

Для показа выбранных свойств в окне настройки нужно установить напротив них признак "Показывать" и нажать кнопку "OK".