Программное обеспечение «Система испытаний авиационных моторов» (СИАМ)

Руководство программиста

Редакция 1.0

2021

АННОТАЦИЯ

Настоящее описание содержит основные сведения об архитектуре, применяемых технологиях при создании программного обеспечения “Система испытаний авиационных моторов” (ПО СИАМ) и описание API для разработки пользовательских расширений на языке Lua.

ПО СИАМ по своему основному назначению является средой разработки специализированного программного обеспечения верхнего уровня автоматизированных систем измерения и управления технологическим процессом испытаний авиационных двигателей определённого типа.

Оглавление

1 Архитектура программного обеспечения

1.1 Состав и функции частей ПО СИАМ

ПО СИАМ состоит из следующих частей:

1) СИАМ-Сервер – сервер каналов реального времени, со следующими функциями:

- владеет контентом испытания и обеспечивает выдачу контекста на АРМ в следующем виде:

а) набор исходных данных (тип двигателя, номер двигателя, вид испытания, этап и др.),

б) прием команд на изменение исходных данных и снятие контрольных точек,

в) контроль ролей и разграничение полномочий (например, если Бригадир уже вошел в систему, то второму Бригадиру в доступе будет отказано; кому-то можно давать команды снятия контрольной точки, а кому-то нет и т.п.),

г) информация о расположении измеренных данных в БДИ;

- сбор данных с измерительных комплексов, приборов;
- расчет контрольных точек из скрипта, созданного в плагине «LuaCalcParams»;
- передача измеренных данных, сигналов, сообщений с Сервера на Клиенты в темпе эксперимента, поддержание синхронизации конфигураций;
- обеспечение сохранения данных в каталоги БДИ в следующем виде:

а) файлы записи измеренных данных в процессе испытания изделия;

б) файлы усреднённых значений параметров на установившихся режимах.

2) СИАМ-Клиент – автоматизированные рабочие места (АРМ) для функционально специализированных по ролям пользователей, со следующими функциями:

- выбор роли пользователя; контроль ролей в пределах всех АРМ осуществляет СИАМ-Сервер;
- обмен с сервером контекстом испытания;
- выбор исходных данных испытаний;
- управление испытанием;
- контроль проведения испытания;
- визуализация хода испытания;
- запуск программы постобработки (СИАМ-отчет);
- выбор и редактирование шаблона протокола испытания;
- просмотр и печать протокола испытания.

3) СИАМ-Отчёт – обеспечивает постобработку зарегистрированных данных, построение отчетов, представление на экране и принтере данных в контрольной точке в виде списка параметров и графиков.

4) ПО БДИ – Программное обеспечение ведения базы данных испытаний, обеспечивает отображение структуры данных, копирование и архивирование результатов испытаний двигателей. Руководство оператора БЛИЖ.409801.100.130-01 34.

5) Редактор шаблона – создание и редактирование элементов отчёта испытаний двигателей, в том числе, в Excel-программе, импорт и экспорт данных шаблона.

1.2 Режимы работы ПО СИАМ

1.2.1 Различаются два основных режима работы ПО СИАМ:

- настройка;
- работа/обработка.

Настройка комплекса ПО разделена на зоны ответственности Администратора и остальных пользователей.

Настройку Сервера производит только Администратор с выделенного рабочего места обслуживания.

Разделение на зоны ответственности ПО СИАМ показано в таблице 1

Таблица 1 – разделение на зоны ответственности при настройке ПО СИАМ

Настройка АРМ клиентов	**Настройка Сервера (Администратор)**
Шаблоны отчетов посредством ПО СИАМ-Отчет или через ПО Редактор шаблонов	Конфигурация Recorder (аппаратная настройка, имена/описание/единицы измерения каналов, калибровки)
	Настройка плагинов
	Настройка скриптов формул расчета
	Настройка формуляров
	Создание ролей, редактирование прав доступа ролей

1.2.2 В ПО СИАМ может выполняться автоматизированный режим реализации управляющих функций, характеризующийся участием человека в принятии решений и (или) их реализации. При этом возможны следующие варианты:

– ручной режим, при котором комплекс технических средств представляет оперативному персоналу контрольно-измерительную информацию о состоянии систем стенда, а выбор и осуществление управляющих воздействий производит человек-оператор;

– режим «советчика», при котором комплекс технических средств вырабатывает рекомендации по управлению, а решение об их использовании принимается и реализуется оперативным персоналом;

1.2.3 В ПО СИАМ может обеспечиваться автоматическое управление оборудованием и технологическим процессом испытательного стенда на следующих этапах испытаний:

– Подготовка испытания.

Проведение диагностики систем и выдача разрешения на проведение испытаний (готовность АСУ ТП).

– Проведение испытания.

Режим нормального функционирования, характеризуемый полной готовностью всей системы для проведения испытаний.

– Обработка и предоставление результатов.

Режим визуализации параметров, построение графиков. Постобработка результатов и печать протокола испытания.

– Аварийный режим.

При возникновении аварийных ситуаций и ошибок в программном обеспечении, диагностические инструменты разрабатываемой АСУ ТП позволяют сохранять набор информации, необходимой для идентификации нештатной ситуации: снимки экранов, текущее состояние памяти, файловой системы.

1.3 Структурная схема ПО СИАМ

1.3.1 На рисунке 1.1 приведена структурная схема организации ПО «СИАМ-сервер»

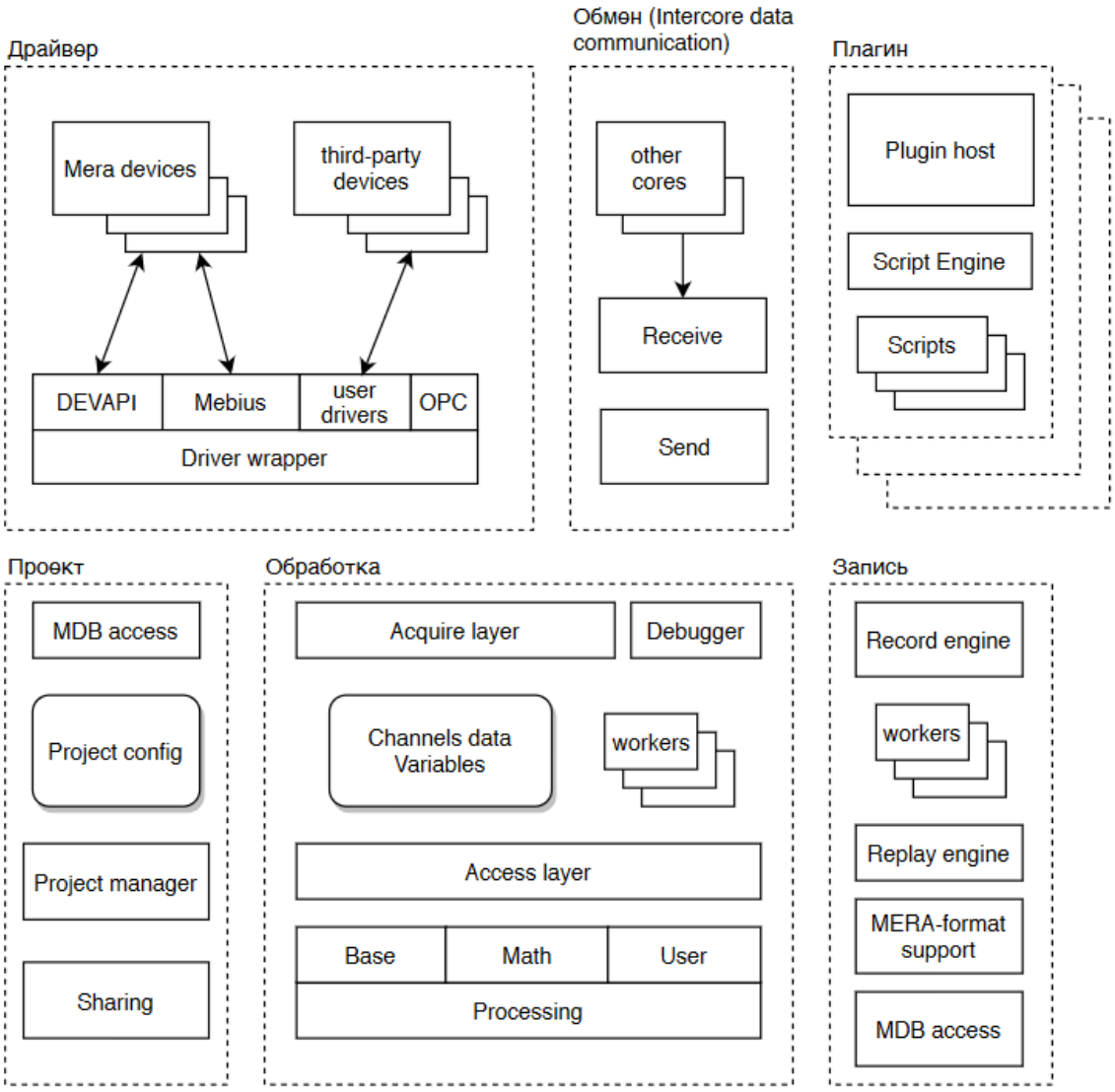


Рисунок 1.1 – структурная схема ПО «Сиам-сервер»

Mera devices – ИВК МЕРА;

Third party devices – пользовательские устройства;

DEVAPI – драйвера для ИВК МЕРА;

Mebius – драйвера для ИВК МЕРА (V2.0);

User Drivers – драйвера для пользовательских устройств;

OPC – сервер для ПЛК;

Driver wrapper – унификация API для всех типов устройств (оболочка);

Other cores – процесс поставщик данных;

Receive – прием обработка данных от источника;

Send – отправка данных потребителей;

Plugin host – сервис–окружение для запуска обработки пользовательских расширений;

Script Engine – программный модуль запуска и обработки пользовательских расширений (скриптов);

Scripts – пользовательские расширения (скрипты);

MDB access – сервис доступа (запись-чтение) к базе данных;

Project config – настройки проекта;

Project manager – программный модуль управления настройками проектов;

Sharing – общие данные (метаданные, GUIID и пр.)

Acquire layer – API для получения данных

Debugger – программный модуль для отладки пользовательских расширений;

Channels data Variables -

Workers – раздельные потоки обработки;

Access layer – API доступа к данным (realtime);

Base – программный модуль базовых операций с данными;

Math – математические вычисления с данными (расширенные);

User – Пользовательские операции;

Processing – программный модуль обработки (для Base, Math,User)

Record engine – программный модуль записи данных;

Replay engine – программный модуль воспроизведения записанных данных;

MERA-format support – унификация API доступа к данным;

2 Средства, используемые при разработке «Системы испытаний авиационных моторов» (ПО СИАМ).

2.1 Применяемые средства хранения исходных кодов

Система хранения исходных кодов: Phabricator (standalone, сервер предприятия);

Система контроля версий: Mercurial;

Средства сборки: MSbuild 2022, Microsoft Visual Studio 2008, C++ Builder 6;

Система управления сборкой: Jenkins CI (standalone, сервер предприятия);

Система хранения артефактов сборки: Sonatype Nexus (standalone, сервер предприятия).

3 Характеристики программного обеспечения

3.1 Разработка пользовательских расширений (скриптов)

3.1.1 Некоторые сведения о языке Lua [1], [2]

Программа на расширяемом языке Lua не имеет понятия основная (ведущая) программа: она работает в среде исполнения, сокращенно называемой “хост”. “Хост-программа” позволяет запускать части кода, написанные на Lua, модифицировать переменные Lua и регистрировать C-функции для использования непосредственно в коде Lua. Благодаря возможности расширения с помощью C-функций, Lua может применяться для решения широкого круга задач.

Лексические соглашения

Именами (идентификаторами) в Lua могут быть любые строки из букв, цифр и символа подчеркивания, не начинающиеся с цифры.

Следующие ключевые слова зарезервированы и не могут быть использованы в именах:

and, break, do, else, elseif, end, false, for, function, if, in, local, nil, not, or, repeat, return, then, true, until, while.

Lua является языком, чувствительным к регистру символов: and – ключевое слово, тогда как And и AND– два разных допустимых идентификатора.

По соглашению, имена, начинающиеся с символа подчеркивания и записанные в верхнем регистре (например _VERSION), зарезервированы для использования в качестве внутренних глобальных переменных, используемых Lua.

Литеральные строки должны быть заключены в одинарные или двойные кавычки и могут содержать C-подобные управляющие последовательности:

\a (bell), \b (backspace), \f (form feed), \n (newline), \r (carriage return), \t (horizontal tab), \v (vertical tab), \\ (backslash), \" (double quote), \' (single quote), и \newline,

(то есть, наклонная черта влево, сопровождаемая реальным newline, который приводит к переводу строки).

Символ в строке может также быть определен числовым значением, через управляющую последовательность \ddd, где ddd последовательность до трех десятичных цифр.

Строки в Lua могут содержать любое 8-разрядное значение, включая вложенные нули, которые могут быть определены как \000.

Литеральные строки могут также быть разграничены парами [[...]]. Литералы в этой форме в скобках могут занимать по несколько строк, содержать вложенные пары скобок [[...]] и не интерпретировать управляющие последовательности. Эта форма особенно удобна для записи строк, которые содержат части программы или другие цитируемые строки. Как пример, в системе использующей ASCII-кодировку, следующие три литерала эквивалентны:

```
"alo\n123\""
'\97lo\10\04923"'
[[alo
123"]]
```

Числовые константы могут быть написаны с факультативной целой частью и тоже факультативными дробными частями. Допустимо применение экспоненциальной формы записи. Примеры имеющих силу числовых констант:

3 3.0 3.1416 314.16e-2 0.31416E1 .34 34. 0xFF 0x100

Lua поддерживает такую удобную вещь, как многократные присваивания.

Следовательно, синтаксис определяет список переменных с левой стороны и список выражений с правой стороны. Элементы в обоих списках отделяются запятыми. Эта инструкция сначала оценивает все значения справа и возможные индексы слева, а затем делает присваивание.

Так, код:

```
i = 3
i, a[i] = 4, 20
```

установит a[3] в 20, но не воздействует на a[4] потому, что i в a[i] оценен прежде, чем ему было присвоено значение 4.

Многократное присваивание может использоваться, чтобы поменять местами два значения, например:

```
x, y = y, x
```

Два списка в многократном присваивании могут иметь различные длины. Перед собственно присваиванием, список значений будет откорректирован к длине списка имеющихся переменных.

Типы и переменные

В языке существуют следующие типы данных:

nil – ничего, обозначает отсутствие какого либо значения,

boolean – булевская переменная, принимает значения **true** либо **false**,

number – числовой тип,

string – строковый тип,

function – функция,

userdata – специальный тип данных, позволяющий хранить в Lua данные из C (фактически это указатель void*),

thread – сопрограмма Lua (позволяет организовать превдо - многопоточность),

table – таблица – ассоциативный массив (набор пар ключ-значение), причем в качестве и ключа и значения может выступать любой тип данных.

Переменные типа **table**, **function**, **thread** и **userdata** не содержат самих данных, в них хранятся только ссылки на соответствующий объект.

В Lua есть три вида переменных: *глобальные*, *локальные* и *поля таблиц*. Любая переменная считается глобальной, если она явно не объявлена как локальная. Локальные переменные существуют в лексическом контексте: локальные переменные доступны функциям, определенным внутри этого контекста.

```
local x, y, z
local a, b, c = 1, 2, 3
local x = x
```

Операторы

В Lua поддерживается в общем стандартный набор операторов, почти как в Pascal или C. Он состоит из операторов присваивания, операторов управления потоком исполнения, вызова функций и описания переменных.

Ниже перечислены основные операции:

– смена знака

+ – * / арифметика

^ возведение в степень

== **~=** равенство

< <= **>** **>=** порядок

not **and** **or** логика

.. конкатенация строк

получение длины строки или массива

Подобно структурам управления, логические операторы рассматривают **nil** как **false** (ложь), а все остальное как истину (**true**).

Оператор конъюнкции **and** вернет **nil**, если первый параметр **nil**, иначе это возвращает второй параметр. Оператор дизъюнкции **or** вернет первый параметр, если он отличается от **nil**, в противном случае это возвращает второй параметр.

Операторы **and** и **or** используют краткое вычисление, то есть второй операнд оценен только в случае необходимости.

Имеются две полезных идиомы в Lua, которые используют логические операторы.

Первая идиома:

```
x = x or v
```

которая является эквивалентной:

```
if x == nil then x = v end
```

Эта идиома устанавливает **x** к значению по умолчанию **v**, когда **x** не установлен.

Вторая идиома такая:

```
x = a and b or c
```

Эта идиома эквивалентна следующему выражению, и является аналогом тернарной операцией сравнения языка C/C++.

```
if a then x = b else x = c end
```

Действует следующая таблица приоритетов и ассоциативности операций:

[справа] **^**

[слева] **not** # **-(unary)**

[слева] ***** **/**

[слева] **+** -

[слева] **<** **>** **<=** **>=** **~=** **==**

[справа] **..**

[слева] **and**

[слева] **or**

В Lua нет выделенной функции, с которой начинается выполнение программы. Интерпретатор последовательно выполняет операторы, которые он получает из файла или от управляющей программы. При этом он предварительно компилирует программу в двоичное представление, которое также может быть сохранено.

Любой блок кода выполняется как анонимная функция, поэтому в нем можно определять локальные переменные и из него можно возвращать значение.

Операторы можно (но не обязательно) разделять символом ';'.

Ниже перечислены основные операторы:

do ... **end**

if ... **then** ... **end**
if ... **then** ... **else** ... **end**
if ... **then** ... **elseif** ... **then** ... **end**
if ... **then** ... **elseif** ... **then** ... **else** ... **end**

while ... **do** ... **end**
repeat ... **until** ...

for var = **start**, **stop do** ... **end**
for var = **start**, **stop**, **step do** ... **end**

return
return ...
break

Блок **do ... end** превращает последовательность операторов в один оператор и открывает новую область видимости, в которой можно определять локальные переменные.

В операторах **if, while** и **repeat** все значения выражения, отличные от **false** и **nil** трактуются как истинные.

Оператор **return** может не содержать возвращаемых значений или содержать одно или несколько выражений (список).

Операторы **return** и **break** должны быть последними операторами в блоке (т.е. должны быть либо последними операторами в блоке кода, либо располагаться непосредственно перед словами **end**, **else**, **elseif**, **until**).

Оператор **for** выполняется для всех значений переменной цикла, начиная от стартового значения и кончая финишным значением, включительно. Третье значение, если оно задано, используется как шаг изменения переменной цикла. Все эти значения должны быть числовыми. Они вычисляются только однажды перед выполнением цикла. Переменная цикла локальна в этом цикле и не доступна вне его тела. Значение переменной цикла нельзя изменять внутри тела цикла. Вот псевдокод, демонстрирующий выполнение оператора **for**:

```
do
local var, _limit, _step = tonumber(start), tonumber(stop), tonumber(step)
if not (var and _limit and _step) then error() end
while (_step>0 and var<=_limit) or (_step<=0 and var>=_limit) do
  ...
  var = var + _step
  end
end
```

Для управления циклами, надо уметь выходить из них по необходимости. Выйти из цикла можно с помощью оператора **break**.

Функции

Определение функции – это исполняемое выражение (конструктор функции), результатом вычисления которого является объект типа функция:

f = **function**(...) ... **end**

В скобках размещается список (возможно пустой) аргументов функции. Список аргументов функции может заканчиваться троеточием – в этом случае функция имеет переменное число аргументов. Между закрывающейся скобкой и оператором **end** размещается тело функции.

В момент выполнения конструктора функции строится также замыкание – таблица всех доступных в функции и внешних по отношению к ней локальных переменных. Если функция передается как возвращаемое значение, то она сохраняет доступ ко всем переменным, входящим в ее замыкание. При каждом выполнения конструктора функции строится новое замыкание.

Вызов функции состоит из ссылки на функцию и заключенного в круглые скобки списка аргументов (возможно пустого). Ссылкой на функцию может быть любое выражение, результатом вычисления которого является функция. Между ссылкой на функцию и открывающейся круглой скобкой не может быть перевода строки.

Если функция принимает единственный аргумент и трактует его как таблицу, элементы которой индексируются именами формальных параметров функции, то в этом случае фактически реализуется вызов механизм поименованных аргументов:

```
function rename( arg )
arg.new = arg.new or arg.old .. ".bak"
return os.rename(arg.old, arg.new)
end
rename{ old = "asd.qwe" }
```

Таблицы

Таблица в Lua – это не только базовый тип данных. И даже не столько. Это фундаментальная основа языка, предопределяющая чуть ли не все возможности Lua.

Таблицы могут использоваться как обычные массивы, таблицы символов, множества, поля записей, деревья. Причем, поскольку функции относятся к значениям первого класса, поля таблицы могут содержать и функции. Таким образом, таблицы могут хранить методы.

Таблица представляет собой ассоциативный массив (набор пар ключ-значение).

Ключём (индексом) и значением может быть любой тип данных, используемых в Lua (за исключением nil).

Конструктор используется для создания таблиц и представляет собой список полей в фигурных скобках.

Поля отделяются друг от друга запятыми. При этом допускается наличие разделителя после последнего поля.

Таблица может быть проинициализирована при создании (стандартный конструктор таблицы):

```
t1 = {} -- пустая таблица
t2 = { 2, 3, 4, "XYZ", 8 } -- обычный массив
t3 = { x = 3, y = "7" } -- таблица с именованными полями «x» и «y»
t4 = { 1, 'line', x = 77 } -- смешанная таблица
t5 = { 1, xxx = 17, } -- разделитель в конце допустим
```

или заполнена позже, инициализируя каждую пару ключ-значение:

```
t4 = {}
t4[1] = 1 -- Значение 1 на место, индексируемое номером 1
t4[2] = 'string' -- Строка 'string' на место, индексируемое номером 2
t4['x'] = 77 -- Число77 на место, индексируемое строкой 'x'
```

Для удобства работы можно вместо индексирования таблицы по имени (строке) использовать это имя как имя поля структуры:

```
t4['x'] = 77
t4.x = 77
```

Чтобы получить обычный массив (таблица t2), просто задаются значения элементов. Ключи будут установлены автоматически.

В Lua обычные массивы индексируются целыми, последовательно-нарастающими числами начиная с единицы.

Пока возможно, Lua внутри себя таблицу хранит как массив, а не как хэш - таблицу.

В этом случае доступ к элементам таблицы происходит почти так же быстро, как в массивах Си. Поэтому без особой нужды не нужно превращать массив в хэш.

Получение размера массива выполняется оператором #:

```
local count = #t
```

Детальную информацию о синтаксисе языка Lua можно найти в [3].

3.1.2 Структура кода и взаимодействие с ПО СИАМ

Поддержка скриптов в ПО СИАМ реализована с помощью плагин расчётных параметров (plgLuaCalcParams).

Скрипт представляет собой программу на Lua [3]. Она может подключать сторонние библиотеки как пользовательские, так и разработанные третьими лицами.

При инициализации ПО СИАМ производится синтаксическая проверка и происходит однократное выполнение программы скрипта. Затем, при переходе системы в режим приема данных, с заданной периодичностью вызывается функция lua_main.

Периодичность устанавливается в окне настроек плагина.

Пример кода:

```
1 --- sample.lua
2
3 counter = 0
4
5 function lua_main()
6   counter = counter + 1
7   setValue("Counter", counter)
8 end
9
10 function foo_prepare()
11   -- Какая-то дополнительная подготовка
12   -- Например загрузка неких коэффициентов из файла
13 end
14
15 foo_prepare()
```

При инициализации системы и при старте будет вызван целиком весь скрипт. Т.е. выполнятся строка 3

```
counter = 0
```

и будет вызвана функция foo_prepare() на строке 15.

При переходе в режим приема данных, периодически будет вызываться lua_main.

Окно настройки параметров скрипта вызывается запуском плагина plgLuaCalcParams на вкладке “Плагины” программы Recorder – рисунок 2.1.

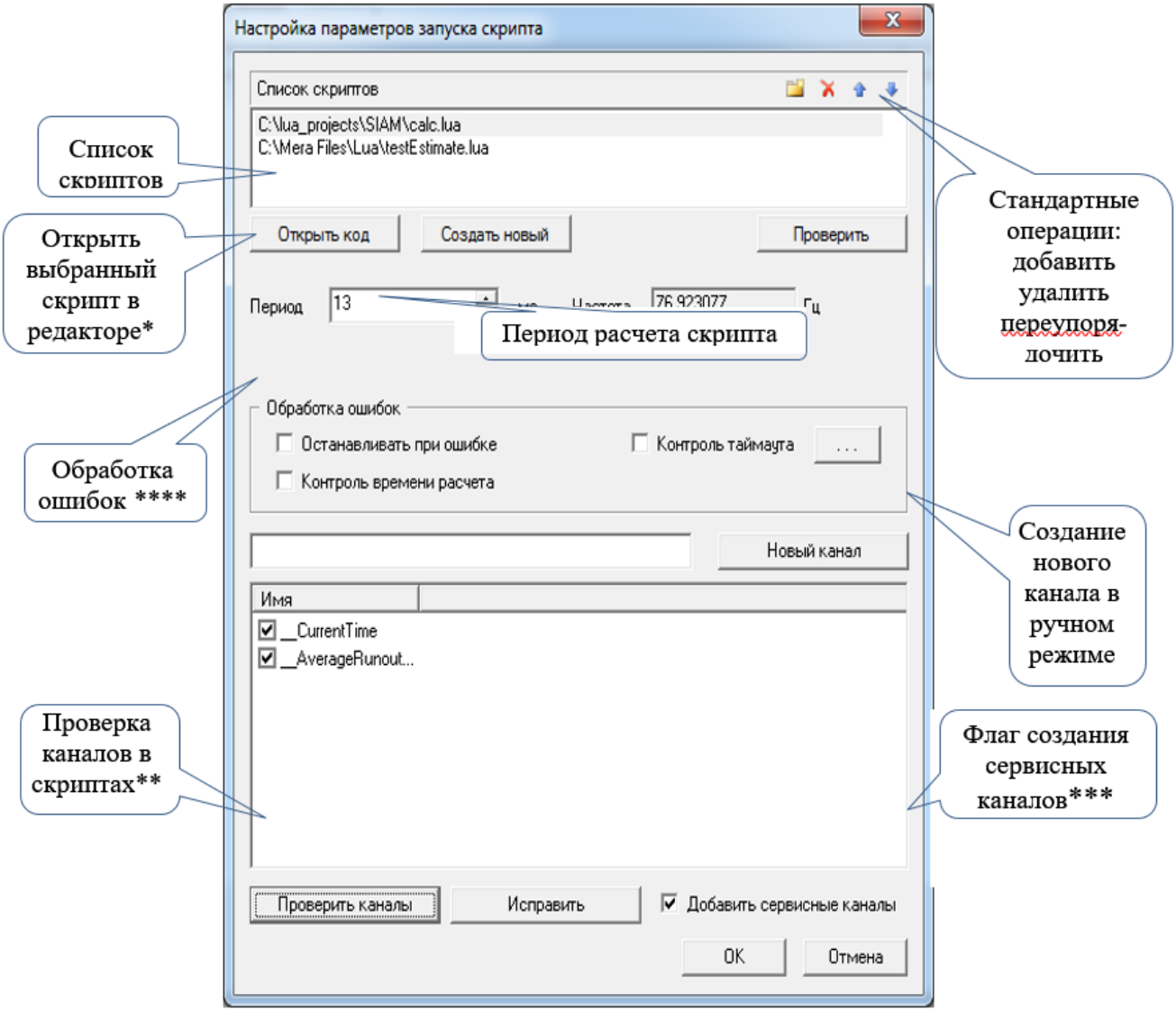


Рисунок 3.1 – Окно “Настройка параметров запуска скрипта”

* Открывается редактор, ассоциированный с расширением .lua

** Находятся выходные каналы в скриптах, проверяется их наличие в программе измерения Recorder’a и пригодность к использованию. «Неисправные» выводятся в список. Их можно изменить после нажатия кнопки «Исправить». Так же будут созданы сервисные каналы метрик исполнения при включении флага (***)

**** Обработка ошибок

– Флаг останавливать при ошибке. Позволяет остановить прием данных в системе (остановка Recorder!) в случае ошибки в одном из скриптов, например при делении на ноль, обращении к не инициализированной переменной или функции.

– Контроль времени расчета. Флаг включающий контроль за временем расчета скрипта и его соответствием заданному периоду вызова этого скрипта. Если программа скрипта выполняется дольше периода её вызова, это считается ошибкой. Реакция зависит от положения предыдущего флага.

– Контроль таймаута. Позволяет избежать зацикливания или зависания программы. Настраивается в количестве выполненных команд. Что позволяет останавливать программу в отладчике без опасения её преждевременного завершения по таймауту.

Функции, доступные из скрипта lua

Стандартные функции ПО Recorder

getValue - Получить текущее значение канала по имени

value:number getValue (name:string)

Входные параметры:

name - строка имя канала

Выходные параметры:

число с плавающей точкой, значение на входе

setValue - Выдать значение в канал по имени

setValue (name:string, value:number)

Входные параметры:

name - строка имя канала

value - число с плавающей точкой, значение

Выходные параметры:

нет

getRef - Получить ссылку на канал по имени

ref:userdata getRef (name:string)

Входные параметры:

name - строка имя канала

Выходные параметры:

ref – ссылка на канал рекордера

Ссылка валидна на протяжении всего времени работы скрипта. Необходимо запрашивать однократно при первом запуске, за пределами функции lua_main. Работа с данными по ссылкам существенно увеличивает скорость взаимодействия с ядром Recorder и может дать эффект при большом количестве используемых каналов при высокой частоте вычислений. При этом использование ссылок немного усложняет код, а встроенное кэширование доступа по именам достаточно эффективно. Поэтому старайтесь использовать данный механизм только по необходимости.

getValueByRef - Получить текущее значение канала по ссылке

value:number getValueByRef (ref:userdata)

Входные параметры:

ref – ссылка на канал рекордера

Выходные параметры:

число с плавающей точкой, значение на входе

setValueByRef - Выдать значение в канал по ссылке

setValueByRef (ref:userdata, value:number)

Входные параметры:

name - строка имя канала

value - число с плавающей точкой, значение

Выходные параметры:

нет

getEstimate - Получить значение скалярной оценки данных канала по имени

value:number, time:number, status:number getEstimate (name:string, estimate: string)

Входные параметры:

name - строка имя канала

estimate - строка название оценки

* - оценка по умолчанию

m – среднее значение (МО)

e – СКО

d - СКЗ

a – амплитуда

r – размах

min – минимальное значение на интервале

max – максимальное значение на интервале

Выходные параметры:

number - число с плавающей точкой, текущее значение оценки

time – штамп времени вычисления оценки

status – состояние канала на момент вычисления оценки

Параметр status может принимать следующие значения:

/// Флаги состояния данных

/// Могут характеризовать как блок данных так и конкретное значение в зависимости от контекста.

/// Состояние делятся на аварийные – когда установлен флаг DATAST_INVALID_DATA и на информационные, в случае если этот флаг сброшен. Например информация о срабатывании уставки.

typedef enum {

/// Корректные данные

DATAST_VALID_DATA = 0x00000000,

DATAST_NULL = 0x00000000,

DATAST_OK = 0x00000000,

/// Состояние - нет данных, начальное состояние

DATAST_EMPTY = 0x80000000,

/// Не корректные данные в общем

DATAST_INVALID_DATA = 0x00000001,

/// Зашкал АЦП

DATAST_ADC_OUT_OF_RANGE = 0x00000002 |

DATAST_INVALID_DATA,

/// Выход за границы ГХ

DATAST_TARE_OUT_OF_RANGE = 0x00000004,

/// Превышение верхней предупредительной уставки

DATAST_ALARM_HI_LOW = 0x00000008,

/// Превышение верхней аварийной уставки

DATAST_ALARM_HI_HI = 0x00000010,

/// Превышение нижней аварийной уставки
DATAST_ALARM_LOW_LOW = 0x00000020,
/// Превышение нижней предупредительной уставки
DATAST_ALARM_LOW_HI = 0x00000040,
/// Обрыв измерительной линии
DATAST_WIRE_BROKEN = 0x00000080 |
DATAST_INVALID_DATA,
/// Отказ оборудования
DATAST_HARDWARE_FAIL = 0x00000100 |
DATAST_INVALID_DATA,
/// Потеря связи с оборудованием
DATAST_LOST_CONNECTION = 0x00000200
DATAST_INVALID_DATA,
/// Канал отключен
DATAST_CHANNEL_DOWN = 0x00000400
DATAST_INVALID_DATA,
/// Данные устарели
DATAST_OBSOLETE_DATA = 0x00000800
DATAST_INVALID_DATA,
/// Короткое замыкание
DATAST_LINE_SHORT_CIRCUIT = 0x00001000
DATAST_INVALID_DATA
} DATAST;

Параметр estimate можно опустить, в этом случае будет использована оценка по умолчанию. Так же можно сохранять только необходимые результирующие параметры, при помощи штатных средств языка.

Например:

v = getEstimate("Имя") --получаем только значение оценки по умолчанию

v,_,s = getEstimate("Имя", "max") – Получаем значение и состояние

_,t,_ = getEstimate("Имя", "*") – время последнего обсчета оценок для канала

getValueEx - Получить текущее значение канала по имени

value:number, time:number, status:number

getValueEx (name:string)

Входные параметры:

name - строка имя канала

Выходные параметры:

number - число с плавающей точкой, текущее значение канала

time – штамп времени значения

status – текущее состояние канала, код состояния аналогично getEstimate

setValueEx - Выдать значение в канал по имени

setValue (name:string, value:number, time:number,

status:number)

Входные параметры:

name - строка имя канала

value - число с плавающей точкой, значение

status – состояние канала, код состояния аналогично getEstimate

Выходные параметры:

нет

getTagFreq - Получить частоту сэмплирования канала по имени

frequency:number getTagFreq (name:string)

Входные параметры:

name - строка имя канала

Входные параметры:

frequency - частота канала

luacpSetTagOpt - Изменить свойство расчетного канала

result:number luacpSetTagOpt (tag:string,

option:string, value:string)

Входные параметры:

tag - строка имени канала

option - строка имени свойства

value - строка значения свойства

Существуют следующие свойства:

auto_put - автоматическая установка последнего известного значения в канал на каждом такте скрипта (для создания канала с постоянной дискретизацией).

Если value равно on , то этот режим включается для данного канала

Входные параметры:

-1 - Неизвестное свойство

0 - свойство изменено успешно

luacpLogMessage - Выдать сообщение в лог

luacpLogMessage (cathegory:string, text:string,

priority:number)

Входные параметры:

cathegory - категория

text - тест сообщения

priority - приоритет сообщения, число от 1 до 7 задает уровень (1 - наименее приоритетный, 7 - наиболее):

1 - Отладочный

2 - Информационный

3 - Уведомление

4 - Уставка предаварийная

5 - Уставка аварийная

6 - Предупреждение

7 - Ошибка

8 - Критический

Выходные параметры:

нет

getRecorderTime - Получить текущее значение счетчика времени в ПО Recorder

time:number getRecorderTime ()

Входные параметры:

нет

Выходные параметры:

число с плавающей точкой, текущее значение времени Recorder

sendRecorderCommand - Выдать команду в ПО Recorder

result:number sendRecorderCommand(command:string, params:string)

Входные параметры:

command - строка команды к ПО Recorder в следующем формате:

rec start - начать запись

rec stop - остановить запись

params - резерв

Выходные параметры:

целое число - результат выполнения команды

-1 - неизвестная команда

-2 - невозможно выполнить команду так как Recorder находится в данный момент в состоянии не позволяющим начать/остановить запись

0 - команда выполнена успешно

getRecorderStatus - Получить текущее состояние ПО Recorder

time:number getRecorderStatus ()

Входные параметры:

нет

Выходные параметры:

целое число, которое может принимать следующие значения:

```
/// Состояния Recorder
enum RECORDER_STATE
{
RS_STOP = 0x00000001,
///< Остановлен
RS_VIEW = 0x00000002,
///< Режим получения данных/просмотра
RS_REC = 0x00000004,
///< Режим записи
RS_PLAYING = 0x00000008,
///< Режим воспроизведения
RS_PLAY = 0x00000008,
///< Режим воспроизведения
RS_HARDWAREFAULT = 0x00000010,
///< Инициализация системы не прошла успешно
RS_INITFAULT = 0x00000020,
///< Инициализация системы не прошла успешно
```

RS_NEEDDEVICERESET = 0x00000040,

///< Требуется Reset девайса

RS_NEEDHARDWARERESET = 0x00000040,

///< Требуется Reset девайса

RS_NEEDSOFTWARERESET = 0x00000080,

///< Требуется програмной части драйвера

RS_NEEDLINKSREFRESH = 0x00000100,

///< Требуется обновление ссылок

RS_PLAYMODE = 0x00000200,

///< Настройка и работа в режиме воспроизведения

RS_SIGNALLOADED = 0x00000400,

///< Конфигурирование прошло успешно

RS_CONFIGCHANGED = 0x00000800,

///< Изменена конфигурация, требуется сохранение

RS_CONFIGMODE = 0x00001000,

///< Режим конфигурирования

RS_PAUSE = 0x00002000,

///< Режим пауза

RS_WAIT_REC_FINISH = 0x00004000,

///< Режим ожидания окончания записи

RS_CALIBRATE_MODE = 0x00008000,

///< Режим калибровки

RS_PACKETLOST = 0x00010000,

///< Потери во время приема

RS_RECEIVEERROR = 0x00020000,

///< Сейчас идут потери

RS_PLAYMODE_ENABLED = 0x00040000,

///< Разрешен режим воспроизведения

RS_ZBALANCE_PROC = 0x00080000,

///< Находимся в процессе балансировки нуля

RS_ERROR_CONDITION = 0x00100000,

///< Находимся в состоянии ошибки

RS_WARNING_CONDITION = 0x00200000,

///< Находимся в состоянии предупреждения

RS_ENABLEDPAUSE = 0x40000000,

RS_TERMINATION = 0x80000000,

///< состояние завершения работы

};

Функции ПО СИАМ (при наличии плагина СИАМ-Сервер)

SIAM_SetDynamicPoint - Установить динамическую точку

SIAM_SetDynamicPoint (time:number)

Входные параметры:

time - вещественное число, время установки точки (указывается в секундах)

Выходные параметры:

нет

Режим точки берется из свойства исходных данных pnt-name-dyn

SIAM_SetDynamicPoint2 - Установить динамическую точку LCPUFT_V_DI

SIAM_SetDynamicPoint2 (time:number, mode:number)

Входные параметры:

time - вещественное число, время установки точки (указывается в секундах)

mode - целое положительное число, числовое значение имени режима

Выходные параметры:

нет

Числовое значение режима должно соответствовать настройке свойства исходных данных pnt-name-dyn

SIAM_SetDynamicPoint3 - Установить динамическую точку CPUFT_I_DDI

SIAM_SetDynamicPoint3 (time1:number, time2:number, mode:number)

Входные параметры:

time1 - вещественное число, время начала точки (указывается в секундах)

time2 - вещественное число, время установки (конца) точки (указывается в секундах)

mode - целое положительное число, числовое значение имени режима

Выходные параметры:

нет

Числовое значение режима должно соответствовать настройке свойства исходных данных pnt-name-dyn

SIAM_StartStaticPoint - Начать усреднение для статической точки

SIAM_StartStaticPoint (duration:number, type:number)

Входные параметры:

duration - вещественное число, длительность усреднения (указывается в секундах)

type - целое число, тип точки

0 - полное измерение

1 - основные параметры

Выходные параметры:

нет

По окончании времени duration установится статическая точка с именем режима взятым из свойства pnt-name на момент окончания усреднения.

SIAM_StopStaticPoint - Остановить усреднение и установить статическую точку

SIAM_StopStaticPoint ()

Входные параметры:

нет

Выходные параметры:

нет

Данная функция используется если усреднение для статической точки нужно остановить раньше положенного времени.

SIAM_GetSourceDataChannel - Получить имя канала в ПО Recorder привязанного к свойству исходных данных

name:string SIAM_GetSourceDataChannel (id:string)

Входные параметры:

id - строка служебного имени свойства исходных данных

Выходные параметры:

'name' - имя канала в ПО Recorder привязанное к данному свойству

SIAM_GetSourceDataType - Получить тип данных свойства исходных данных

type:number SIAM_GetSourceDataType (id:string)

Входные параметры:

id - строка служебного имени свойства исходных данных

Выходные параметры:

'type' - целое число, тип данных свойства

VT_I4 (3) - целый

VT_R8 (5) - вещественный

VT_ARRAY (8192) - список

VT_BSTR (8) - строка

SIAM_GetSourceDataStringValue - Получить значение свойства исходных данных в виде строки

value:string SIAM_GetSourceDataStringValue (id:string)

Входные параметры:

id - строка служебного имени свойства исходных данных

Выходные параметры:

value - строка значения данного свойства

Если свойство типа список, то вернется строковое представление свойства. В остальных случаях, если свойство не строкового типа, то значение преобразуется в строку.

SIAM_GetSourceDataIntValue - Получить значение свойства исходных данных в виде целого числа

value:number SIAM_GetSourceDataIntValue (id:string)

Входные параметры:

id - строка служебного имени свойства исходных данных

Выходные параметры:

value - целое число, значения данного свойства

Если свойство типа список , то вернется численное представление свойства. В остальных случаях, если свойство не целого типа, то возвращается 0.

SIAM_GetSourceDataDoubleValue - Получить значение свойства исходных данных в виде вещественного числа LCPUFT_D_S

value:number

SIAM_GetSourceDataDoubleValue(id:string)

Входные параметры:

id - строка служебного имени свойства исходных данных

Выходные параметры:

value - вещественное число, значения данного свойства

Если свойство не вещественного типа, то возвращается 0.

SIAM_SetSourceDataStringValue - Изменить значение строкового свойства исходных данных

SIAM_SetSourceDataStringValue(id:string, value:string)

Входные параметры:

id - строка служебного имени свойства исходных данных

value - строка значения свойства

Выходные параметры:

нет

Если свойство типа список , то меняется как строковое представление свойства так и численное. В случае невозможности определить численное значение, оно присваивается -1. В остальных случаях, если свойство не строкового типа, то команда не выполняется.

SIAM_SetSourceDataIntValue - Изменить значение целочисленного свойства исходных данных

SIAM_SetSourceDataIntValue(id:string, value:number)

Входные параметры:

id - строка служебного имени свойства исходных данных

value - целое число, значение свойства

Выходные параметры:

нет

Если свойство типа список , то меняется как строковое представление свойства так и численное. В случае невозможности определить строковое значение, оно присваивается пустой строке. В остальных случаях если свойство не целого типа, то команда не выполняется.

SIAM_SetSourceDataDoubleValue - Изменить значение вещественного свойства исходных данных LCPUFT_V_SD

SIAM_SetSourceDataDoubleValue(id:string, value:number)

Входные параметры:

id - строка служебного имени свойства исходных данных

value - вещественное число, значение свойства

Выходные параметры:

нет

Если свойство не вещественного типа, то команда не выполняется.

Функции ПКПАС (при наличии плагина ПКПАС)

E_Alarm - Показать аварийное сообщение ПКПАС

E_Alarm(text:string, msgid:number, tableid:number)

Входные параметры:

id - тест сообщения

msgid - целое число, идентификатор сообщения

tableid - целое число, идентификатор таблицы в которой нужно показать данное сообщение

Выходные параметры:

нет

PE_Alarm – Показать предаварийное(предупредительное) сообщение ПКПАС

PE_Alarm(text:string, msgid:number, tableid:number)

Входные параметры:

id - тест сообщения

msgid - целое число, идентификатор сообщения

tableid - целое число, идентификатор таблицы в которой нужно показать данное сообщение

Выходные параметры:

нет

Delete_Alarm - Удалить сообщение ПКПАС

Delete_Alarm(msgid:number)

Входные параметры:

msgid - целое число, идентификатор сообщения

Выходные параметры:

нет

4 Обращение к программному обеспечению

4.1 Исходные данные испытаний двигателя

4.1.1 В качестве исходных данных испытаний авиационного двигателя представляется набор свойств процесса испытаний, отражающих специфику названий объекта испытаний, этапов, видов и режимов испытаний, путь в базе данных испытаний к файлам регистрации параметров объекта испытаний в виде непрерывных значений параметров и в виде среза значений набора параметров.

4.1.2 Список свойств процесса испытаний создаётся в подпрограмме СИАМ-Сервер – рисунок 4.1.

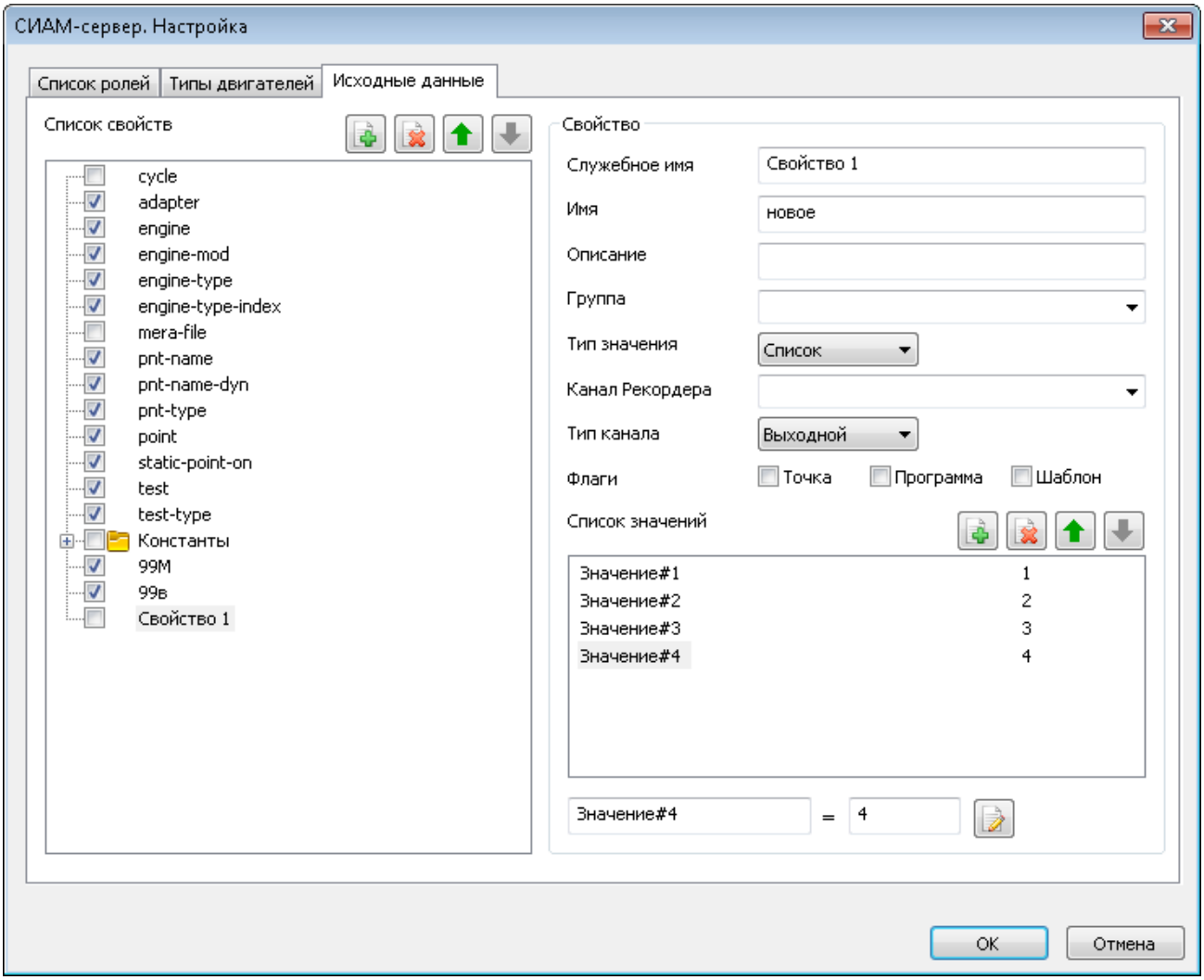


Рисунок 4.1 – Вкладка “Исходные данные” подпрограммы СИАМ-Сервер

4.1.3 Список свойств можно изменять добавлением новых свойств и удалением существующих. Некоторые свойства, обязательные для работы СИАМ, удалить невозможно.

Добавляемое свойство описывается в области “Свойство”.

Свойство имеет служебное имя (на английском языке) для использования в программах и имя для использования в текстовых документах.

Тип значения свойства может быть целым, вещественным, строкой, списком.

Канал Рекордера выбирается для того, чтобы обмениваться по нему значениями свойства. Направление обмена выбирается в “Тип канала”: Входной, Выходной, Вход/выход.

В том случае, когда типом значений свойства является список значений, как показано на рисунке 4.2, его значения создаются в поле “Список значений”.

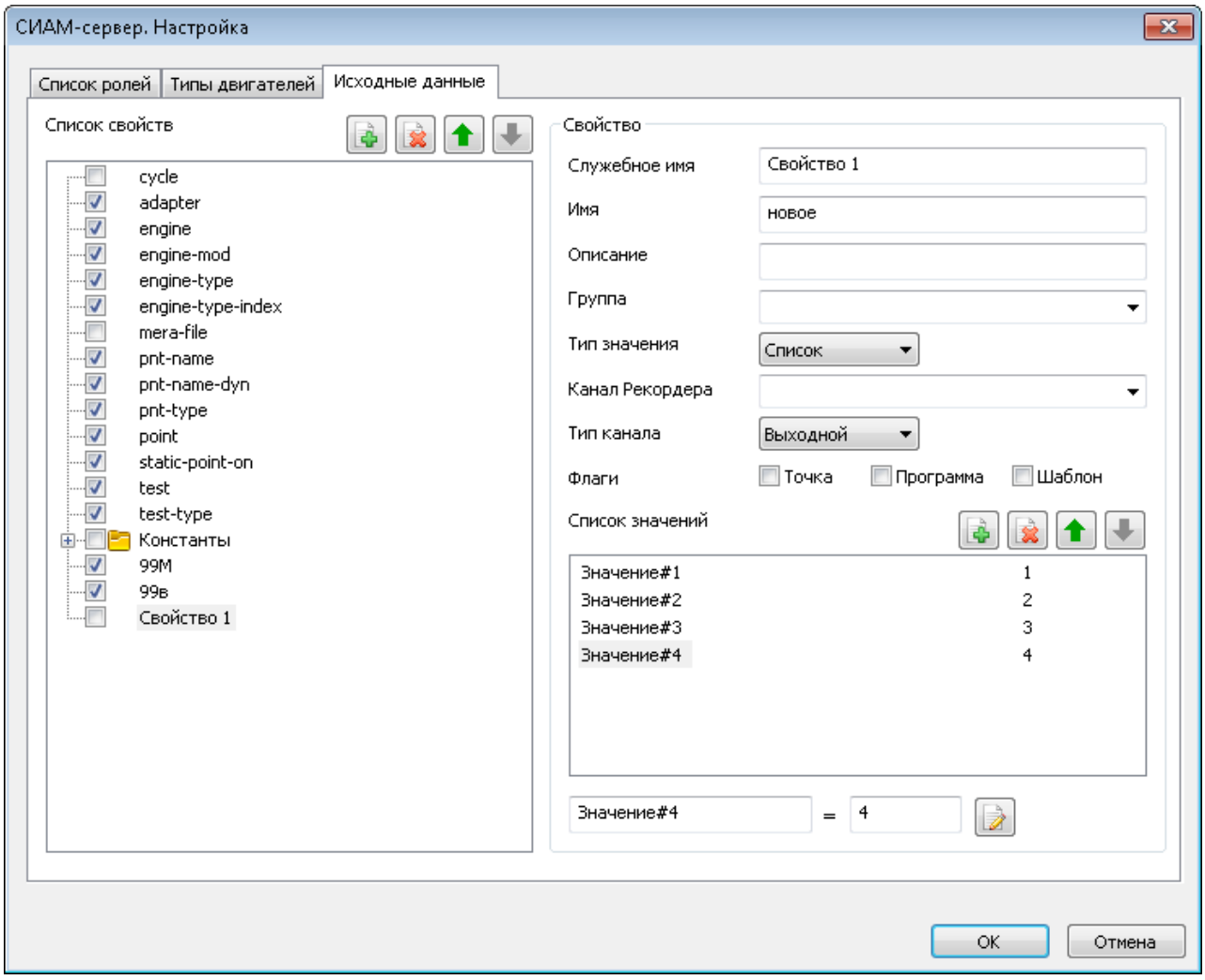


Рисунок 4.2 – Добавление в исходные данные нового свойства

Список значений имеет строковые и числовые значения, которые на равных условиях используются в тексте программ.

4.1.4 Испытываемый двигатель отмечается следующими свойствами:

– engine-type / тип двигателя, например, ПС-90А,

ПД-14;

– engine-type-index / индекс типа двигателя, служит для присвоения модификации двигателя числового значения;

– engine-mod / модификация двигателя, ПС-90

ПС-90А

ПС-90А-76;

– engine / номер двигателя;

– adapter / например, Адаптер 1;

– test-type / вид испытаний, например, ПИ,

ПСИ,

ПП,

ППСИ;

– test / название этапа, например, Консервация,

Расконсервация,

Приработка,

Отладка запусков,

Акт сдачи;

– pnt-name / название режима, например, МГ,

0,37НОМ,

0,7НОМ,

НОМИНАЛ,

ВЗЛЁТНЫЙ;

– pnt-name-dyn / название динамического режима, например, ХП,

ЛЗ,

ЗАПУСК,

САР_МГ;

– point / номер точки, сквозной для всех режимов испытаний у данного номера двигателя;

– pnt-type / тип статической точки, Полное измерение,

Основные параметры;

– static-point-on / установка статической точки, “1” – ставить,

“0” – не ставить;

– cycle / номер цикла в циклическом испытании.

4.1.5 Свойства можно объединять в группы – рисунок 4.3.

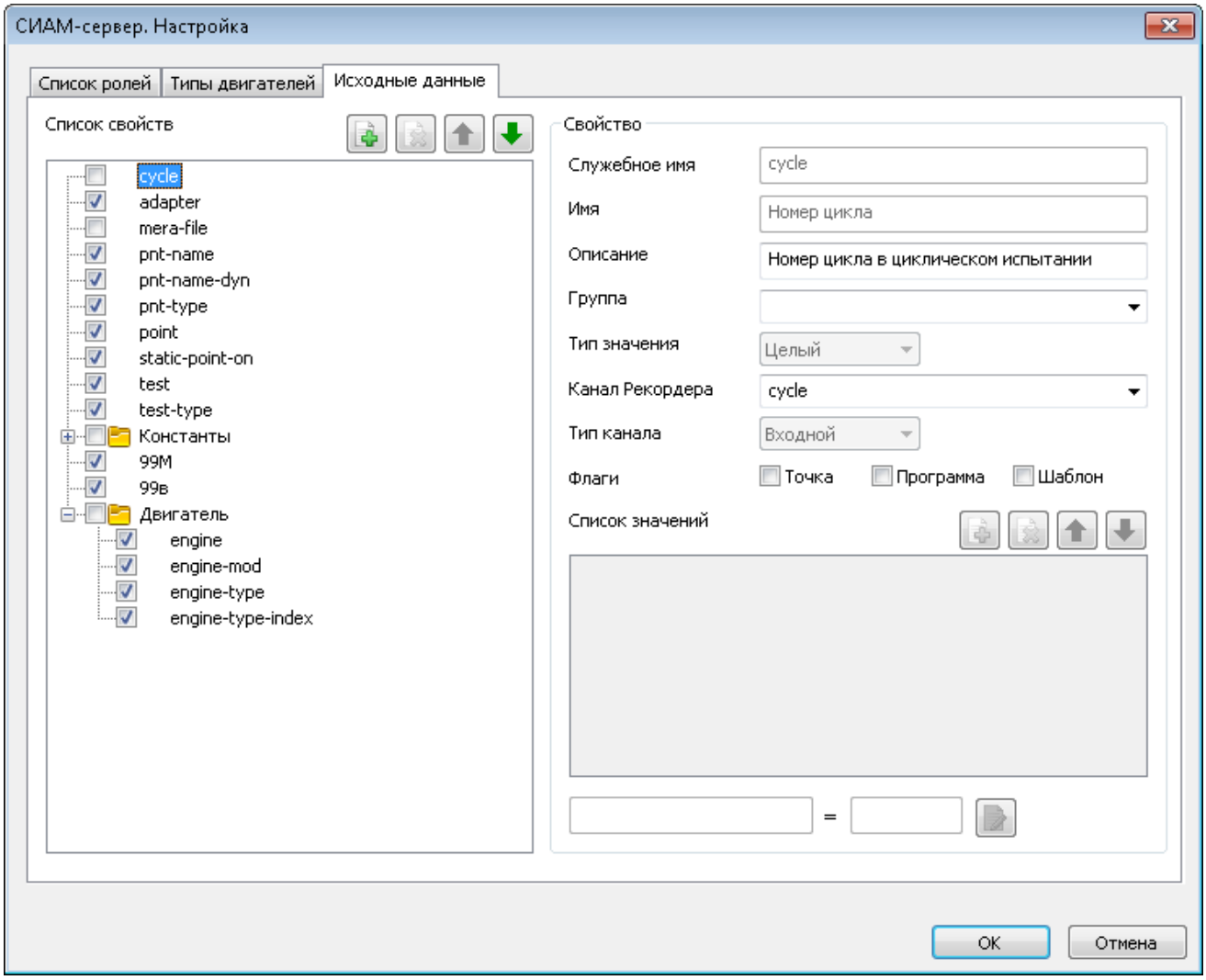


Рисунок 4.3 – Создание группы “Двигатель” в списке свойств

4.1.6 Набор свойств процесса испытаний и значения зарегистрированных параметров объекта испытаний хранятся в файле, расположенном по указанному пути в базе данных испытаний:

mdb.1900\mdbu\ПС-90А\7291028\ПИ\Поверочное испытание \USTX\0183.ustx, как “data” и как “value” соответственно – рисунок 4.4.

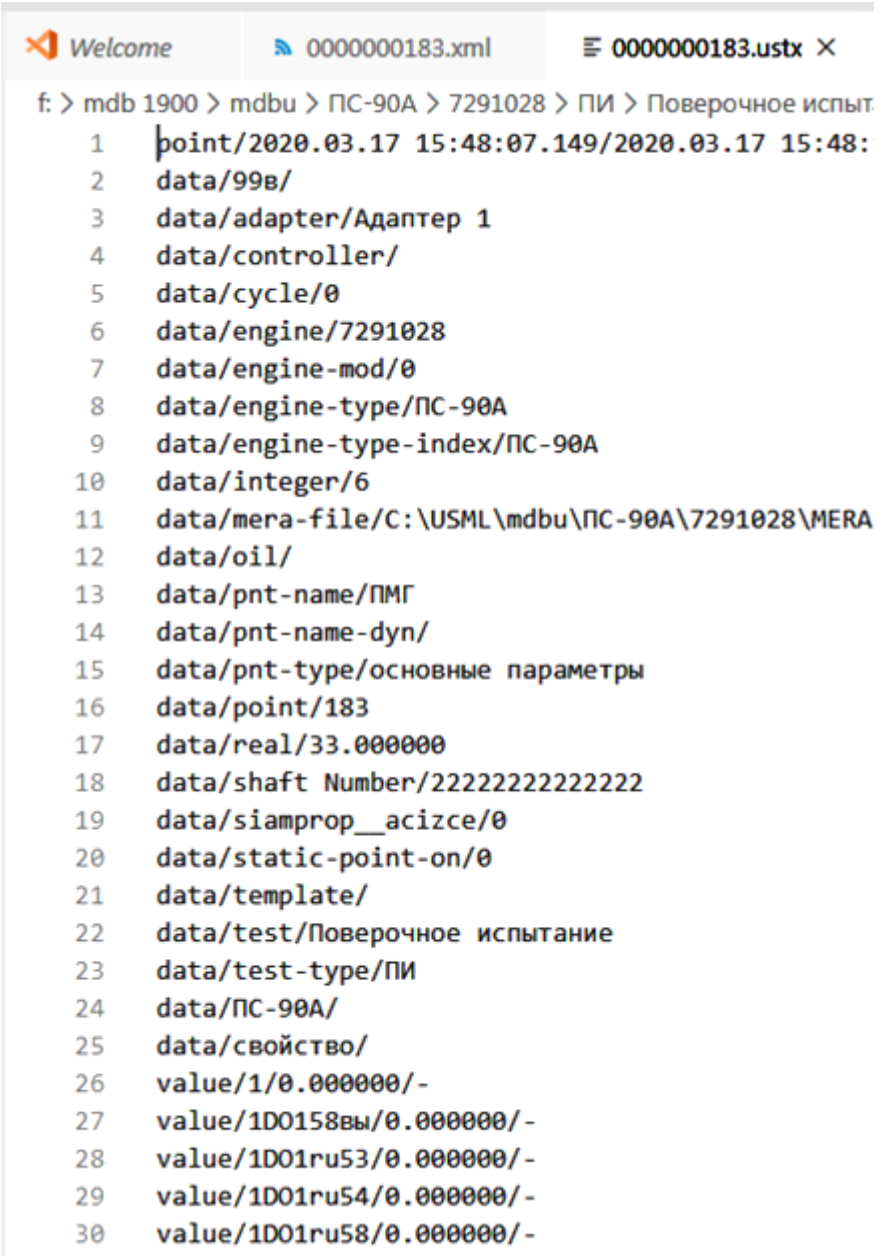


Рисунок 4.4 – Файл стационарной точки среза

4.1.7 Выбор исходных данных из созданных свойств испытания двигателя в процессе испытания двигателя выполняется в Клиенте – рисунок 4.5,

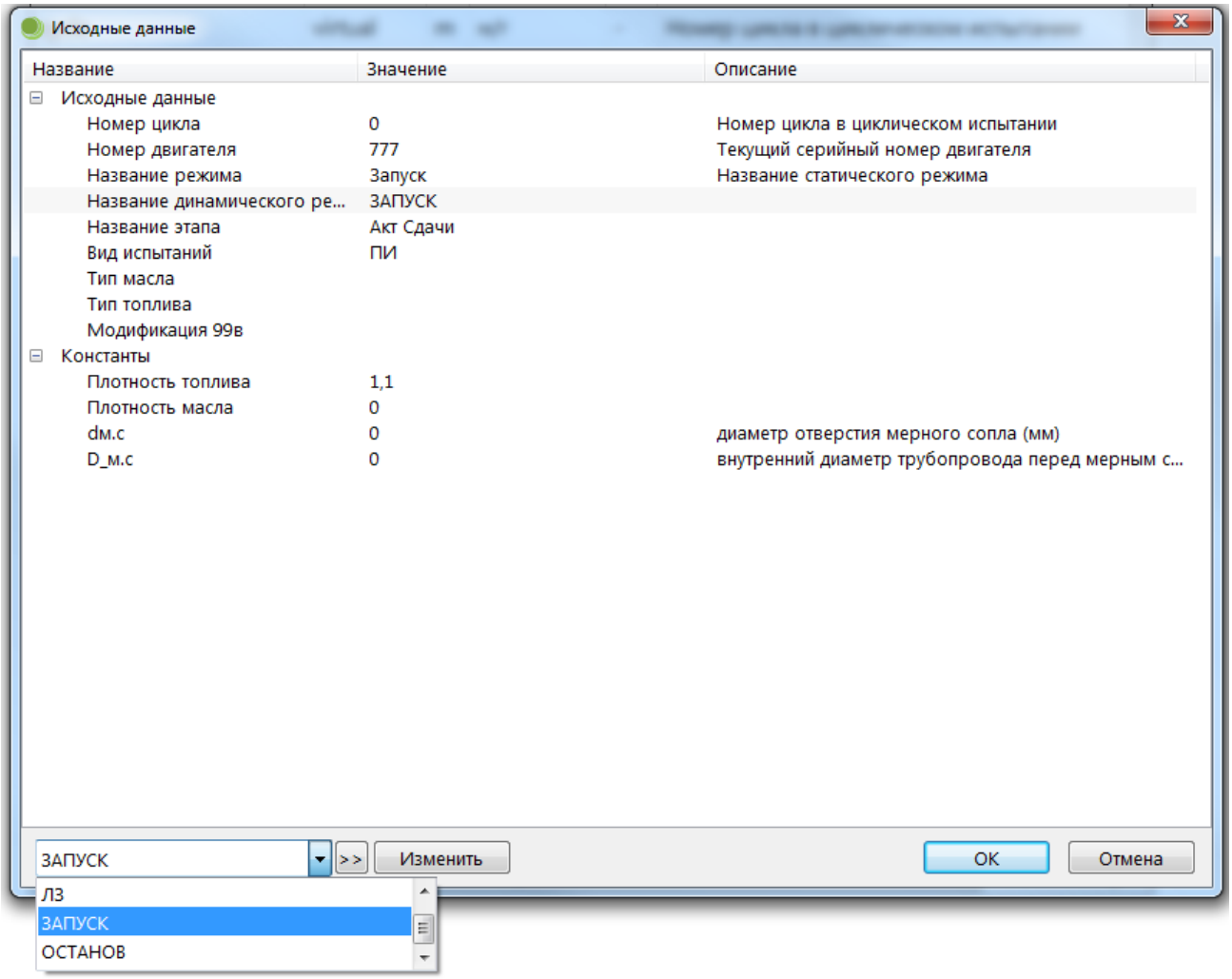


Рисунок 4.5 – Выбор исходных данных в Клиенте

и излагается в руководстве оператора специализированного программного обеспечения СИАМ конкретного стенда.

4.2 Создание экранных мнемосхем

4.2.1 На панели главного окна ПО Recorder кнопкой  вызывается окно “Формуляры” – рисунок 4.6.

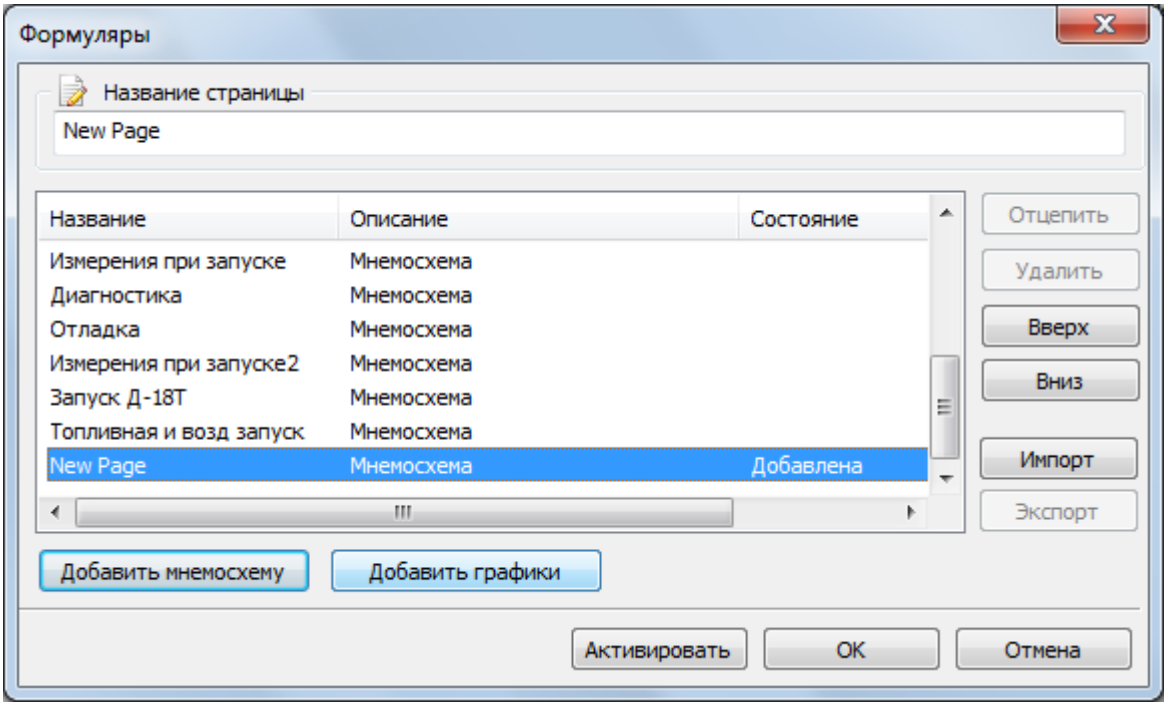


Рисунок 4.6 – Окно настройки формуляров для отображения данных

На СИАМ-Сервере это окно можно использовать для создания новых формуляров и пользовательских страниц (см. Руководство пользователя БЛИЖ.409801.005-01 90).

4.2.2 На панели основного окна ПО Recorder кнопкой  вызывается список для выбора режимов отображения боковой панели и список всех формуляров для выбора отображений в основном окне – рисунок 4.7.

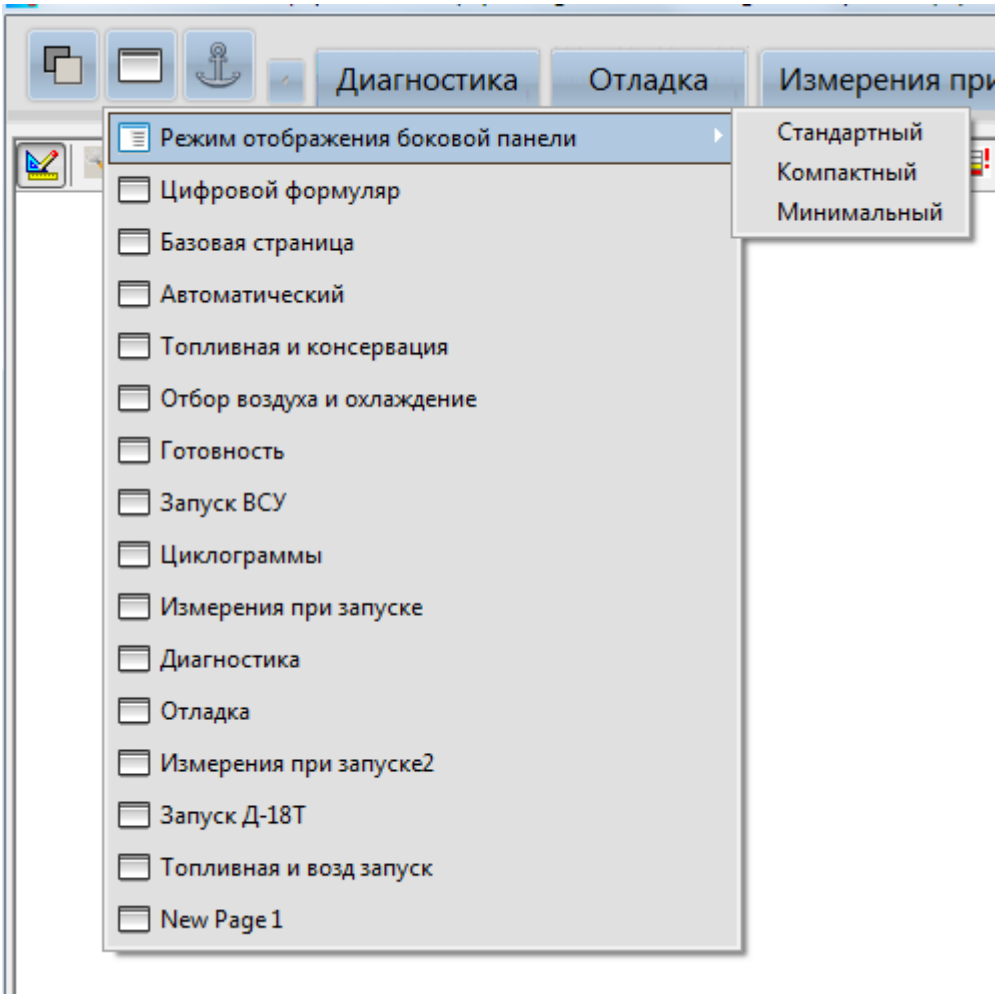


Рисунок 4.7 – Список для выбора режимов отображения основного окна ПО Recorder

4.2.3 Для создания элементов мнемосхем используется панель мнемосхемы ПО Recorder, которая открывается и скрывается нажатием комбинации Ctrl клавиатуры и левой клавишей мыши на свободной области мнемосхемы – рисунок 4.8.

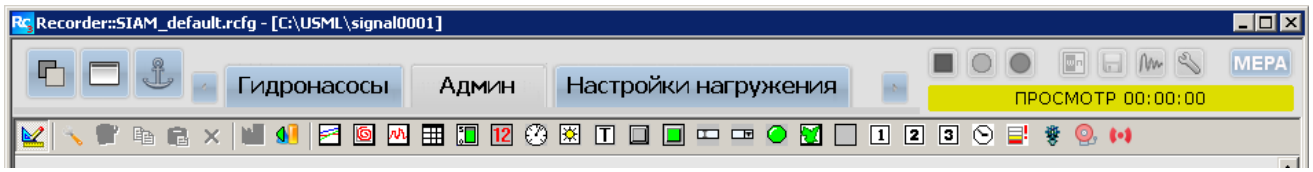


Рисунок 4.8 – Панель создания мнемосхемы

4.2.4 При групповом выделении элементов мнемосхемы – рисунок 4.9, выравнивание элементов с помощью панели внизу экрана будет происходить по ключевому элементу, имеющему широкую рамку выделения – рисунок 4.10. Ключевым элементом становится элемент, перемещённый на передний план.

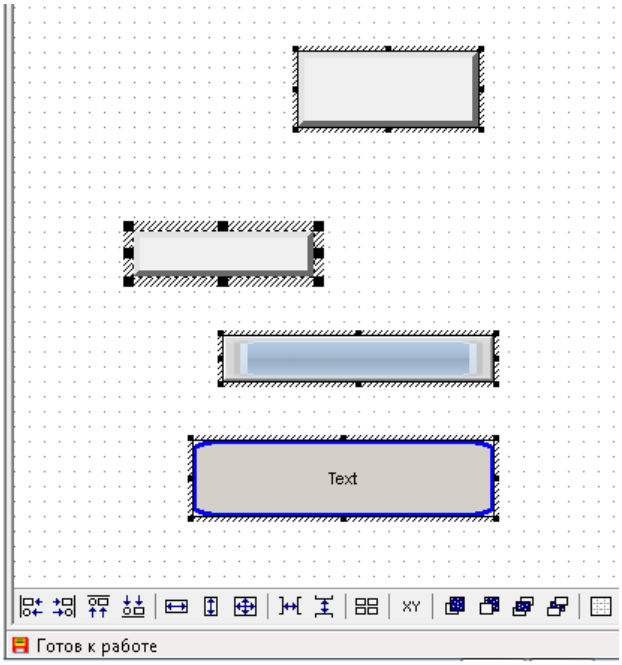


Рисунок 4.9 – Групповое выделение элементов мнемосхемы

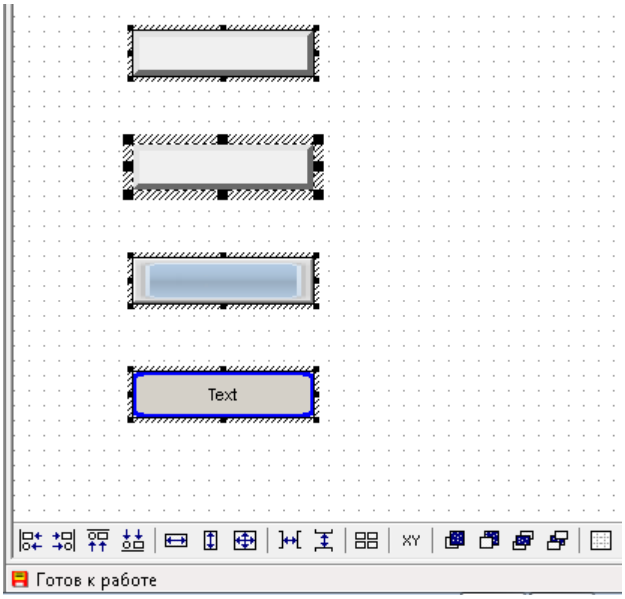


Рисунок 4.10 – Выравнивание элементов мнемосхемы

4.2.5 Кнопками 1 , 2 и 3 из панели настройки мнемосхемы возможно создать вкладку с мнемосхемой управления испытаниями – рисунок 4.11.

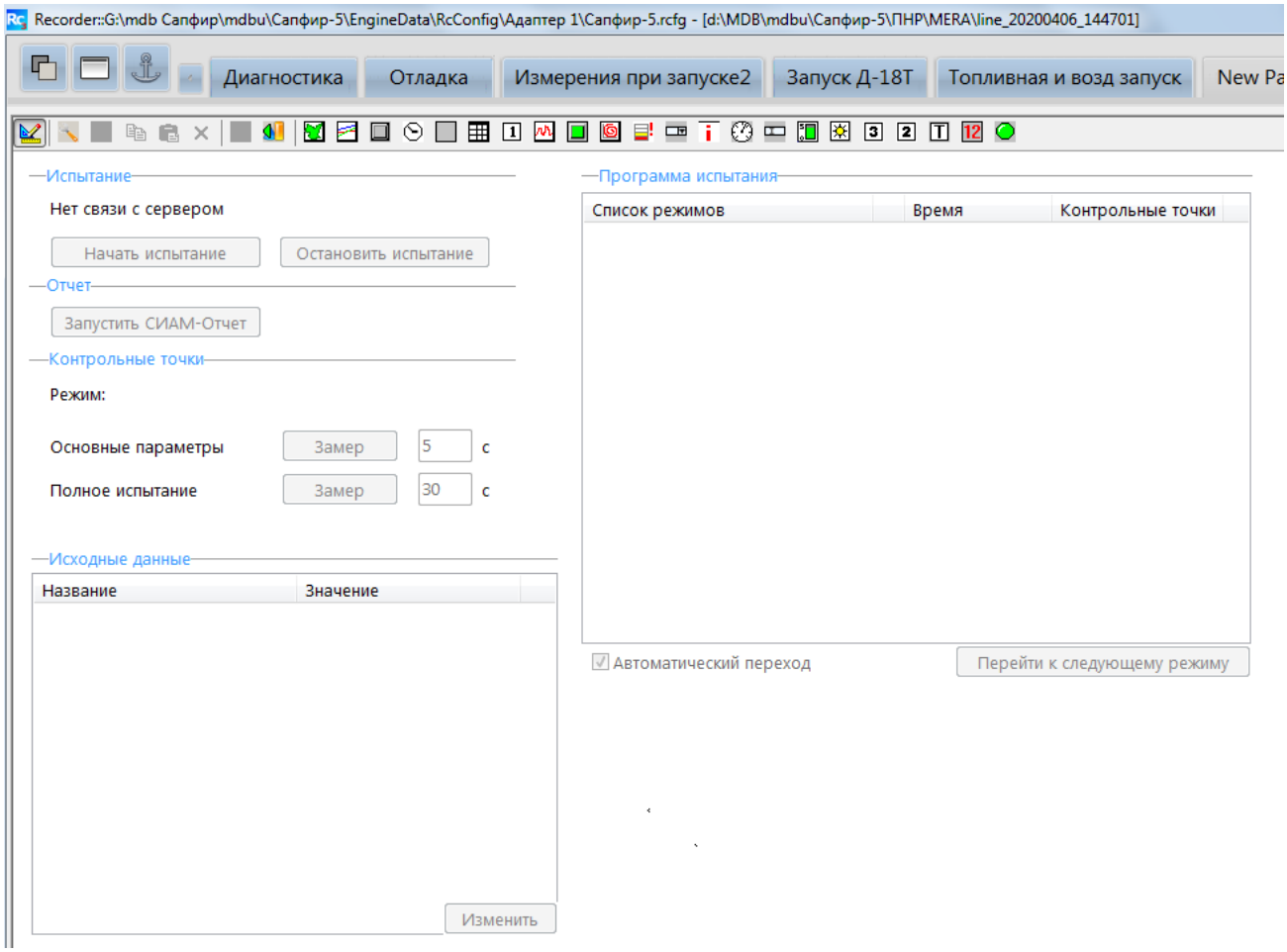


Рисунок 4.11 – Формуляр управления испытаниями

На клиенте поля “Исходные данные” и “Программа испытаний” будут заполнены данными, созданными и выбранными из исходных данных испытаний.

4.2.6 Добавлением элемента “Текущее время” из панели настройки мнемосхемы в выбранном формуляре создаётся панель отображения текущего времени – рисунок – 4.12.

Рисунок 4.12 – Панель “Текущее время”

Выбор в вызываемом “Свойства” типе компонента “Таймер” – рисунок 4.13,

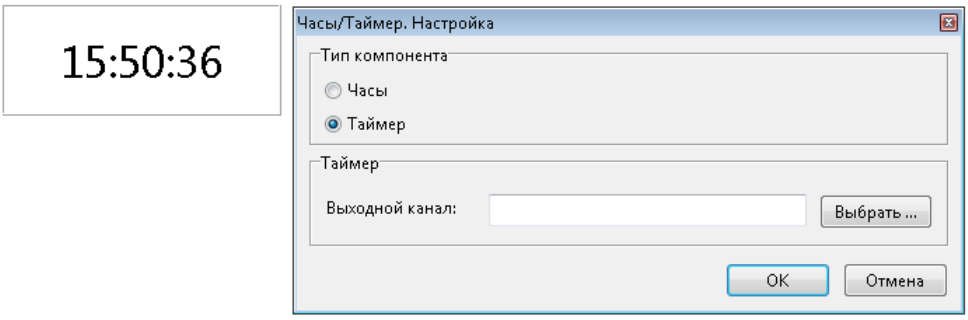


Рисунок 4.13 – Выбор панели “Таймер”

приводит к появлению панели “Таймер” – рисунок 4.14.

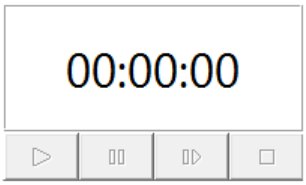


Рисунок 4.14 – Панель “Таймер”

Кнопки “Пуск”, “Временный останов”, “Продолжение”, “Останов” панели “Таймер” активируются в режимах “ПРОСМОТР” и “ЗАПИСЬ” ПО Recorder – рисунок 3.15.

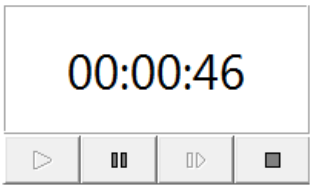


Рисунок 4.15 – Таймер включён

При переходе в режим “ОСТАНОВЛЕН” ПО Recorder значение панели “Таймер” обнуляется.

4.2.7 Добавлением  создаётся элемент управления “Кнопка”.

Типы кнопок выбираются из выпадающего списка – рисунок 4.16.

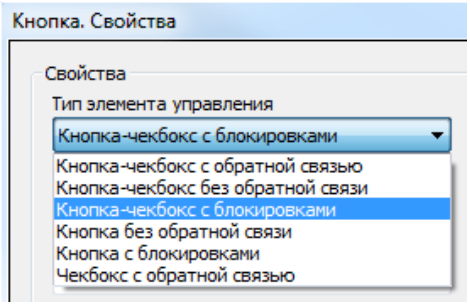


Рисунок 4.16 – Типы кнопок

Возможные настройки свойств кнопки показаны на рисунке 4.17.

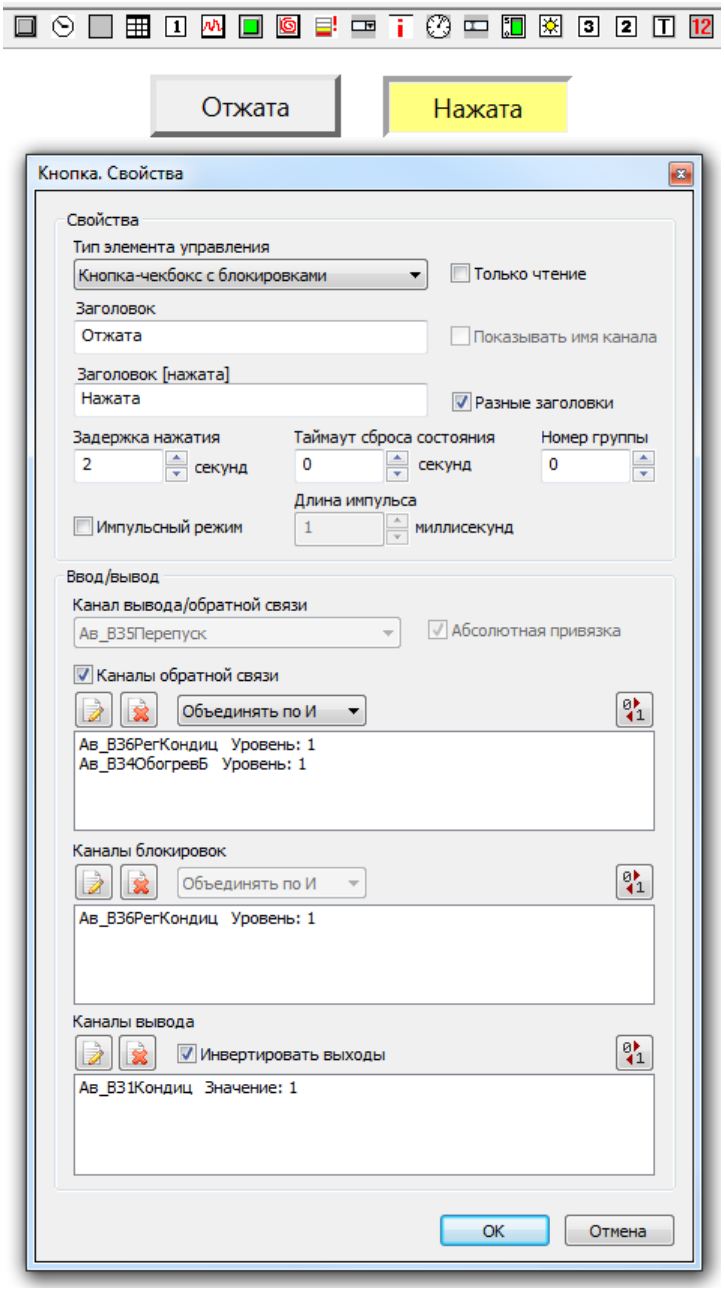


Рисунок 4.17 – Свойства кнопки

Для обратной связи, как для статуса переключения, можно использовать как канал вывода, так и отдельные каналы. Если используется несколько каналов обратной связи, необходимо указывать логику их объединения.

Каналы блокировок используют как условие доступности переключения.

Задержка нажатия и таймаут сброса состояния используют как защиту от случайных нажатий.

Импульсный режим используют, когда необходимо только временное воздействие сигнала.

Настройки внешнего вида показаны на рисунке 4.18.

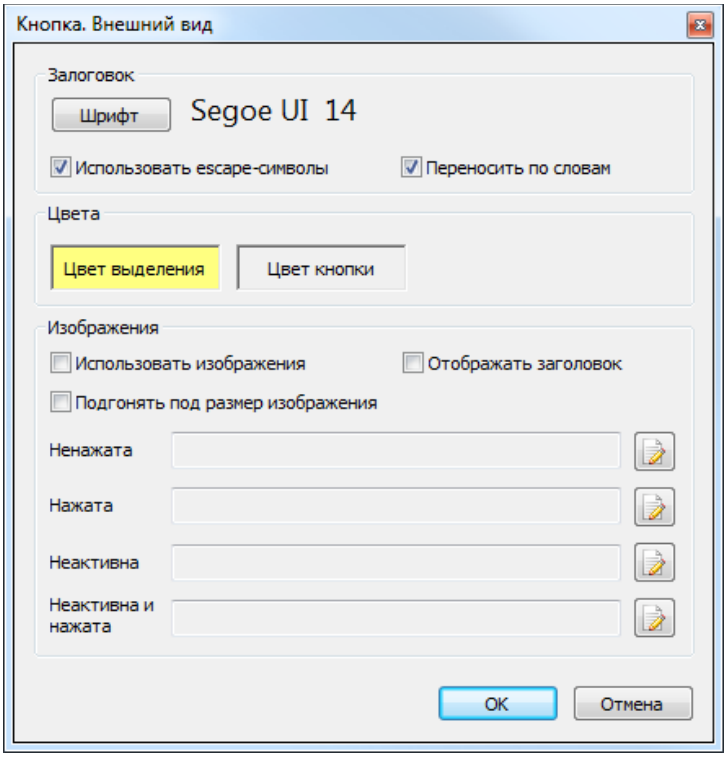


Рисунок 4.18 – Настройки внешнего вида кнопки

Пример использования изображения показан на рисунке 4.19.

Рисунок 4.19 – Тумблер в выключенном и включённым положении

4.2.8 Добавлением  в выбранном формуляре создаётся параметрический график. Настройка параметрического графика заключается в выполнении следующих процедур.

В окне “Настройка” ПО Recorder на вкладке “Каналы” создаётся параметрический канал, например канал зависимости тяги двигателя от скорости вращения ротора КВД – рисунок 4.20.

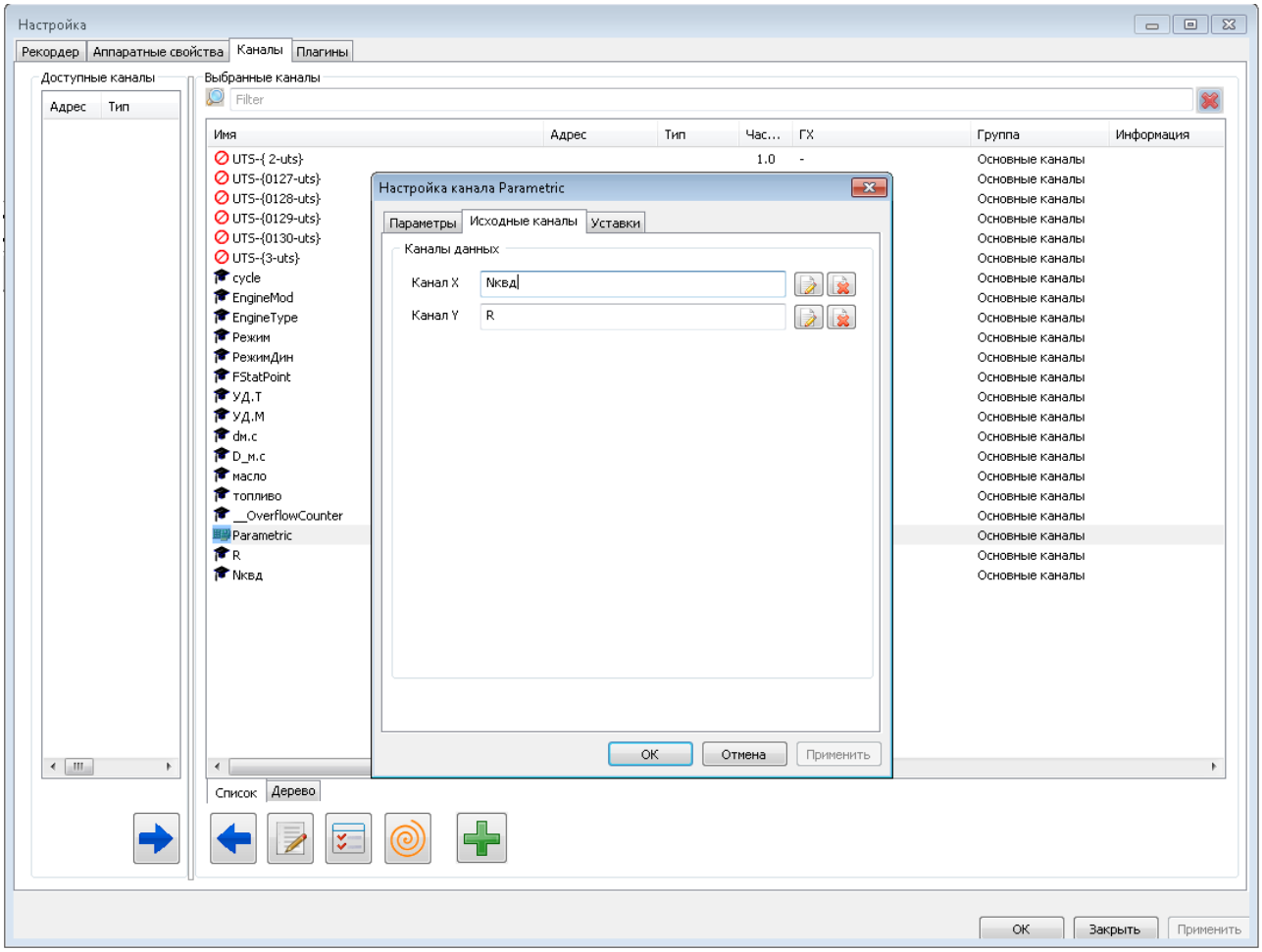


Рисунок 4.20 – Создание параметрического канала

В окне “Настройка параметрического графика”, вызываемого через “Свойство” добавленного параметрического графика, создаётся линия допусков параметрического графика – рисунок 4.21:

а) добавляется линия “noname” допусков,

б) выбирается “Параметрический канал”,

в) ставится флажок “Отображать линии допусков”,

г) назначаются диапазоны осей.

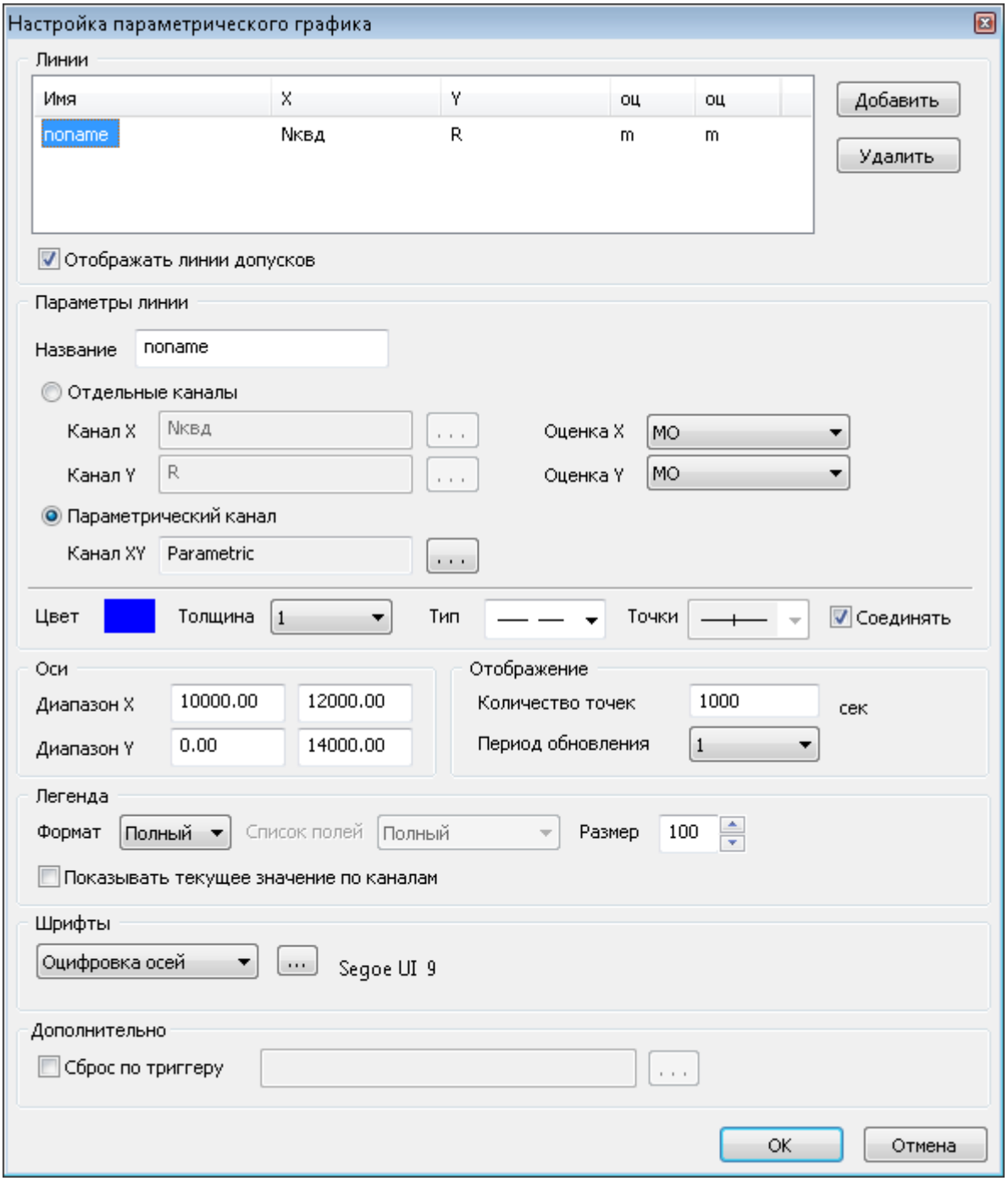


Рисунок 4.21 – Создание линии допусков параметрического графика

В окне “Настройка канала Parametric”, вызываемого через “Свойство“ канала, на вкладке “Уставки” – рисунок 4.22:

а) ставится флажок “Вкл Верхняя аварийная”,

б) с помощью  вызывается окно “Настройка линии уставки”,

в) вводятся и добавляются значения “X” и “Y” линии уставки.

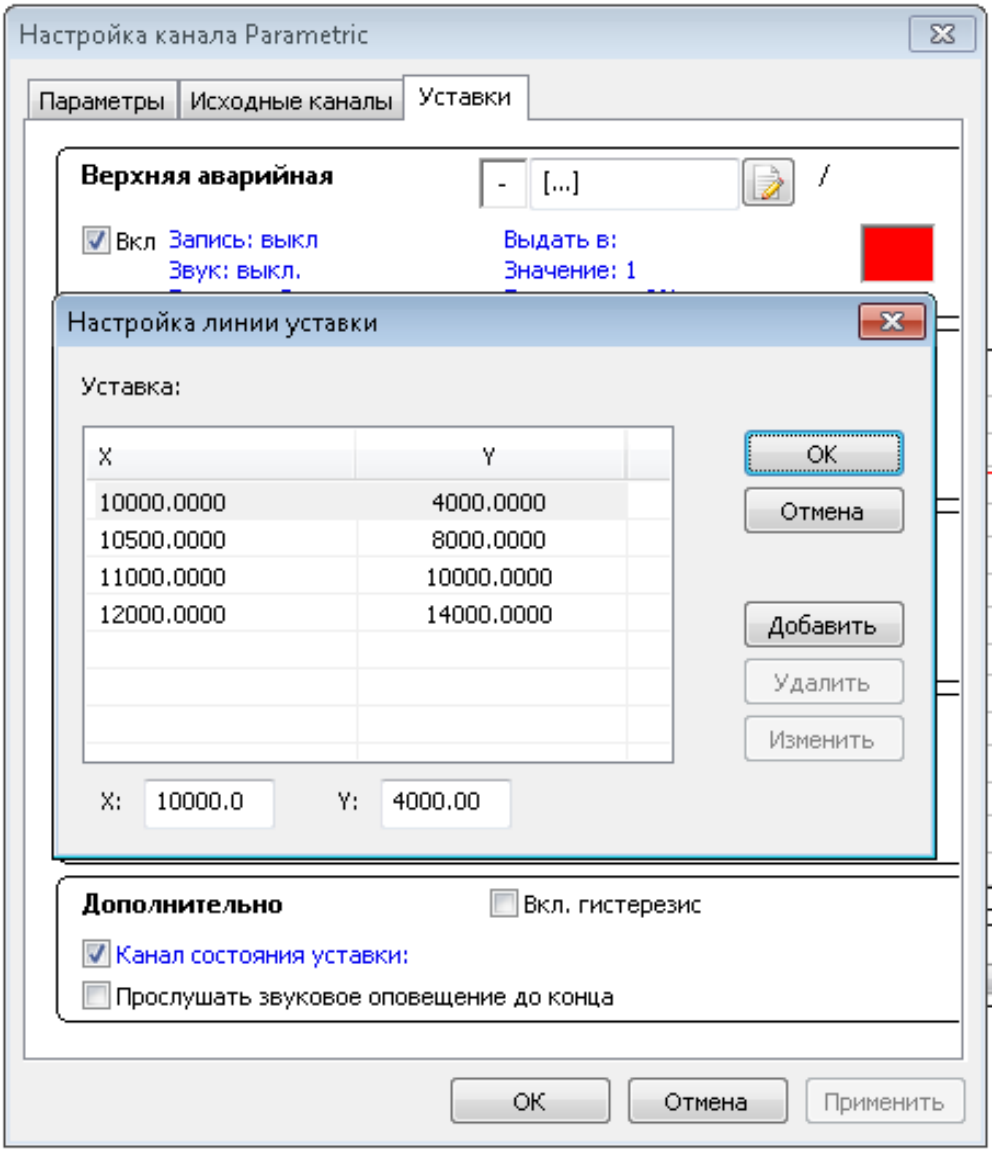


Рисунок 4.22 – Настройка линии параметрической уставки

В результате на параметрическом графике появляется линия параметрической уставки – рисунок 4.23.

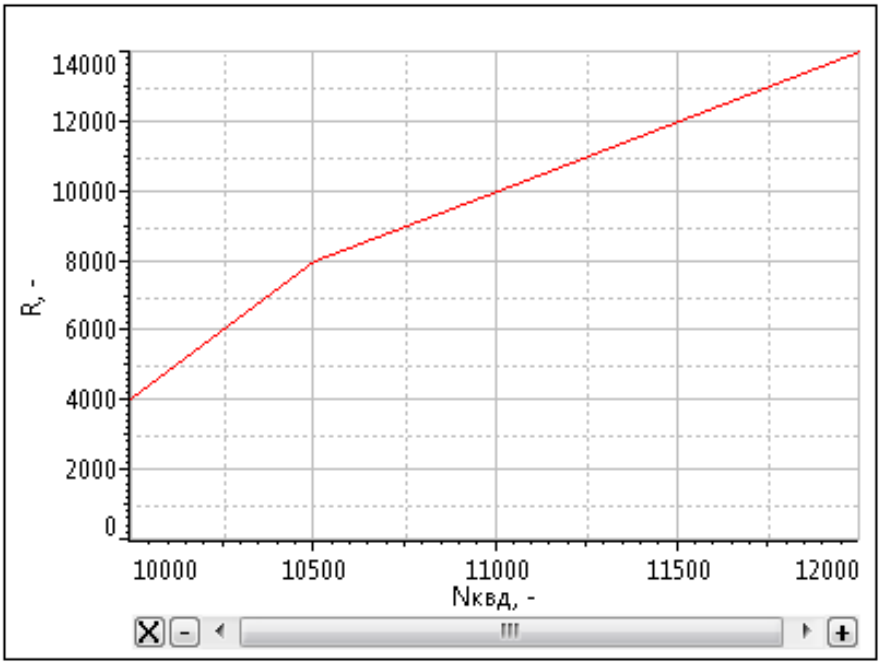


Рисунок 4.23 – Линия параметрической уставки

На параметрическом графике можно создать и уставку по уровню параметра – рисунок 4.24.

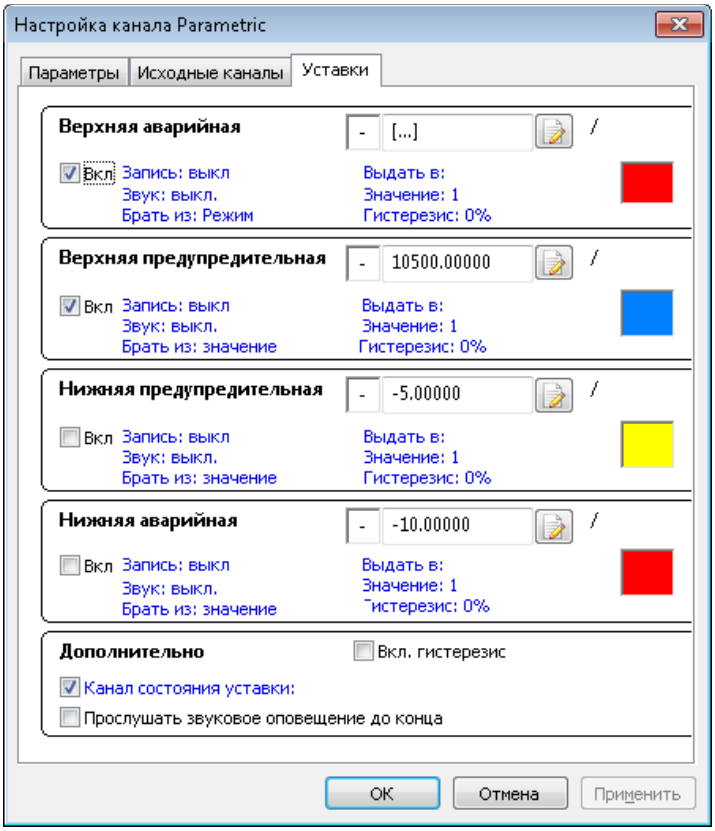


Рисунок 4.24 – Создание уставки по уровню параметра

В результате на графике появятся два вида уставок – рисунок 4.25.

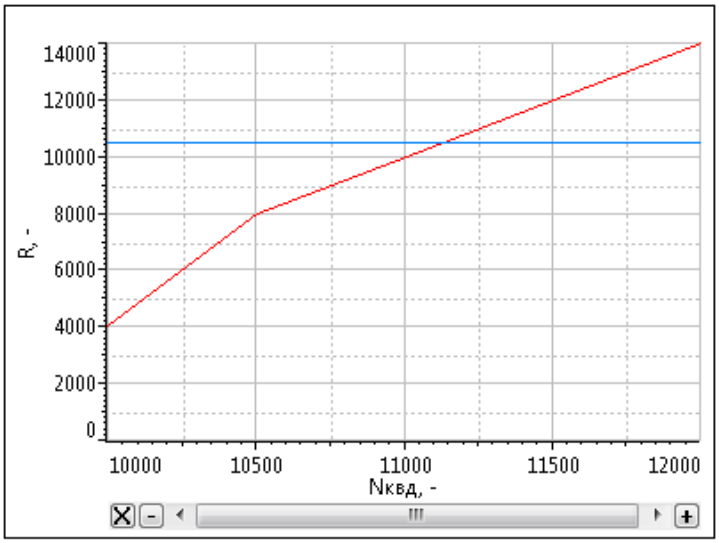


Рисунок 4.25 – Два вида уставок

4.2.9 Добавлением создаётся элемент мнемосхемы “Лампочка”. В свойствах этого элемента – рисунок 4.26,

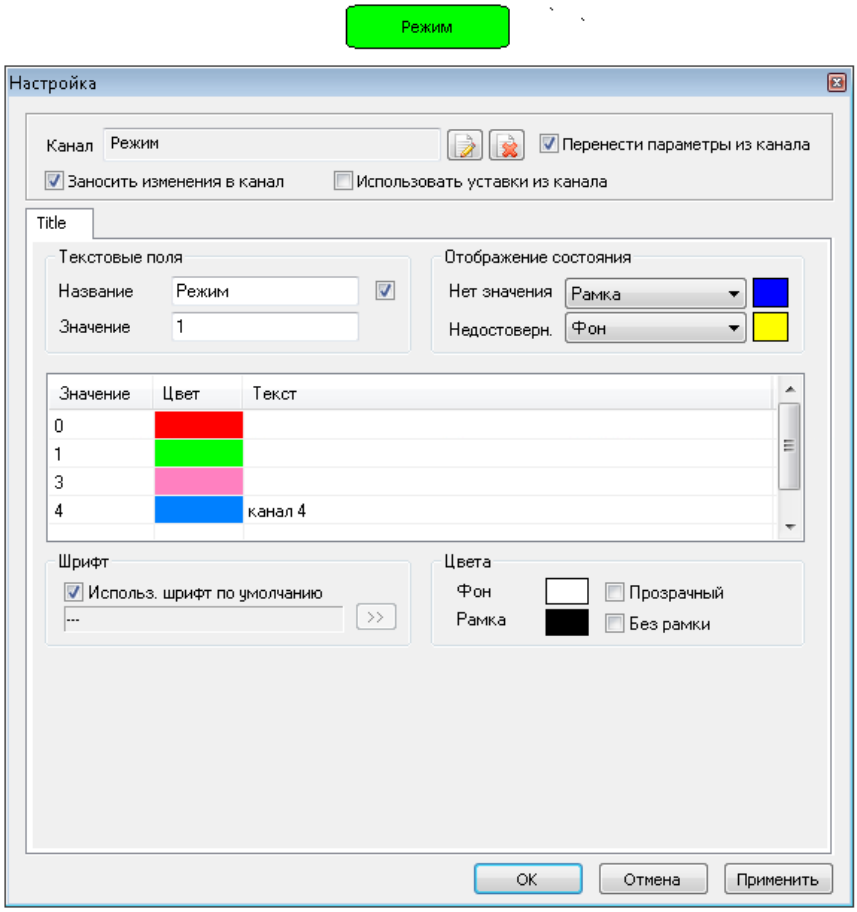


Рисунок 4.26 – Окно настроек элемента “Лампочка”

создаётся необходимое количество значений индикатора по цвету и тексту.

Созданные значения заносятся в указанный канал, при установки флажка “Заносить изменения в канал” и извлекаются из канала, при установке флажка “Перенести параметры из канала”.

4.2.10 Добавлением элемента  “Комбинированный список” из панели настройки мнемосхемы в выбранном формуляре создаётся панель отображения списка значений данных для передачи по выбранному каналу.

Настройка в свойствах панели заключается в выборе канала вход/выход передачи данных, установке количества строк списка, выборе каналов блокировок и их значений в виде “0” и ”1”.

После ввода необходимых значений окно настройки приобретает вид – рисунок 4.27.

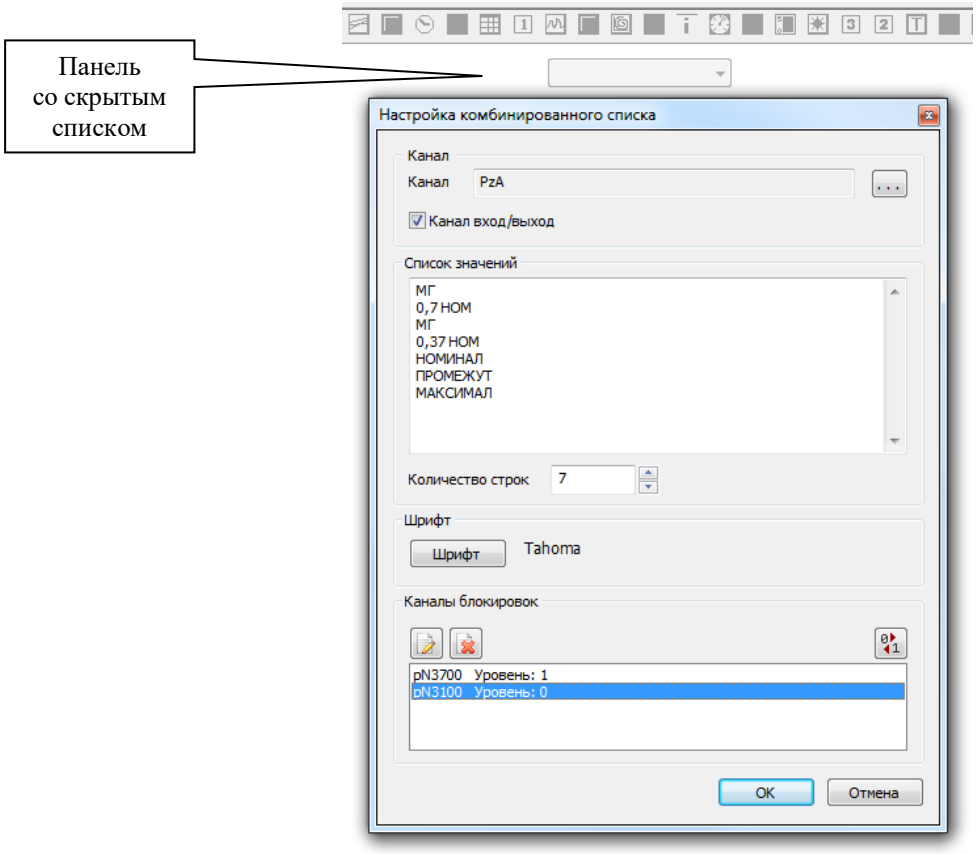
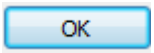


Рисунок 4.27 – Настройка комбинированного списка

После нажатия  на экране остаётся панель со скрытым списком.

4.2.11 Добавление элемента  “Индикатор аварийных ситуаций” из панели настройки мнемосхемы в выбранном формуляре создаётся поле, в котором соответствующим скриптом отображается состояние процесса испытания двигателя – рисунок 4.28.

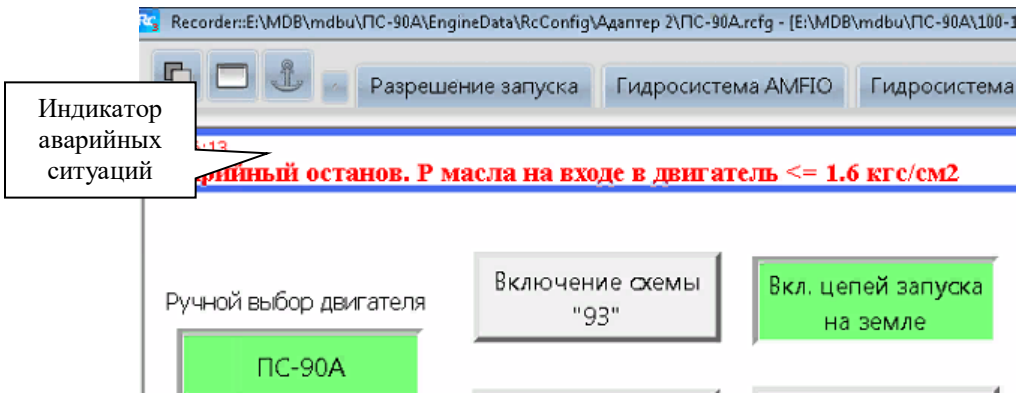


Рисунок 4.28 – Индикатор аварийных ситуаций

4.2.12 Добавлением  на мнемосхеме создаётся журнал событий – рисунок 4.29.

#	Дата/Время	Приори...	Категория	Событие
	25.02.2022 1...	Уведом...	Контроль ПО	Проверка целостнос
	25.02.2022 1...	Предуп...	Контроль ПО	Обнаружен поврежд
	25.02.2022 1...	Уведом...	Контроль ПО	Проверка целостнос
	25.02.2022 1...	Инфор...	Recorder	Core loading started...
	25.02.2022 1...	Инфор...	Recorder	core module loaded
	25.02.2022 1...	Инфор...	Recorder	core module was co
	25.02.2022 1...	Инфор...	Recorder	Default UI server sele
	25.02.2022 1...	Инфор...	Recorder	UI server loaded
	25.02.2022 1...	Инфор...	Recorder	UI class was created
	25.02.2022 1...	Инфор...	Recorder	Инициализация встр
	25.02.2022 1...	Инфор...	Recorder	Загрузка ресурсов
	25.02.2022 1...	Инфор...	Recorder	Инициализация мене
	25.02.2022 1...	Инфор...	Recorder	Создание панели упр
	25.02.2022 1...	Инфор...	Recorder	Инициализация встр
	25.02.2022 1...	Инфор...	Recorder	UI class initialized
	25.02.2022 1...	Инфор...	Recorder	#Start at Fri Feb 25 14
	25.02.2022 1...	Инфор...	Recorder	#version: 3.3.1.63a
	25.02.2022 1...	Уведом...	Recorder	****** Начало проце

Рисунок 4.29 – Журнал событий

Нажатием право кнопки мыши в поле журнала вызывается меню выбора вида журнала – рисунок 4.30.

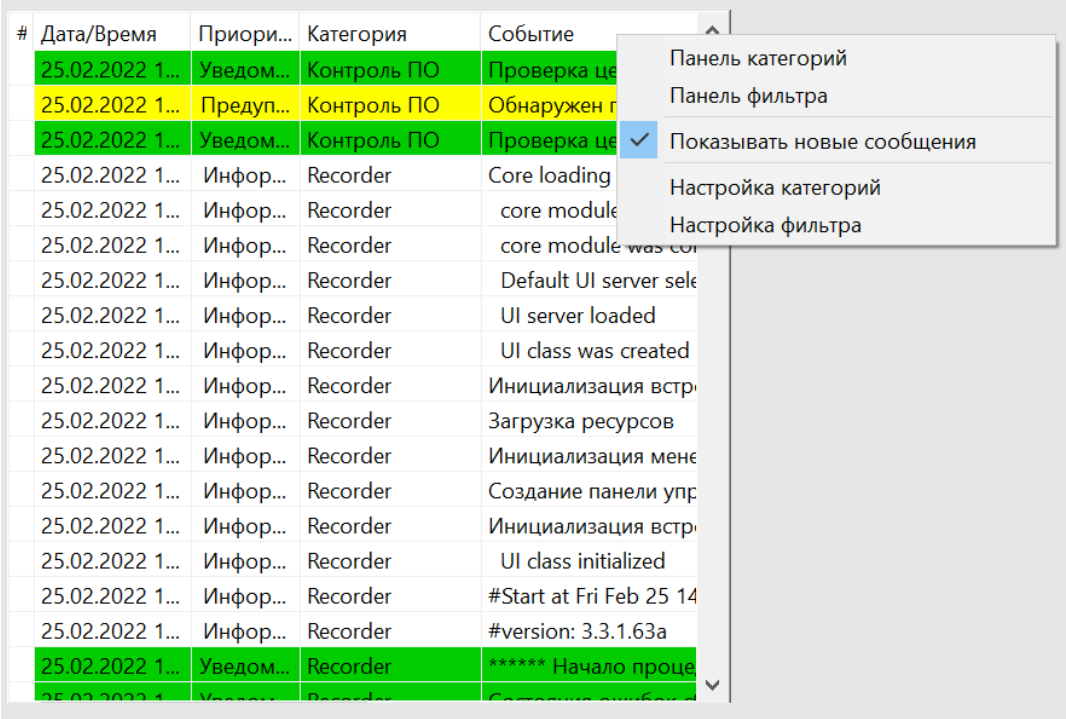


Рисунок 4.30 – Меню вида журнала событий

После выбора панелей и кнопок настройки журнал может иметь следующий вид – рисунок 4.31.

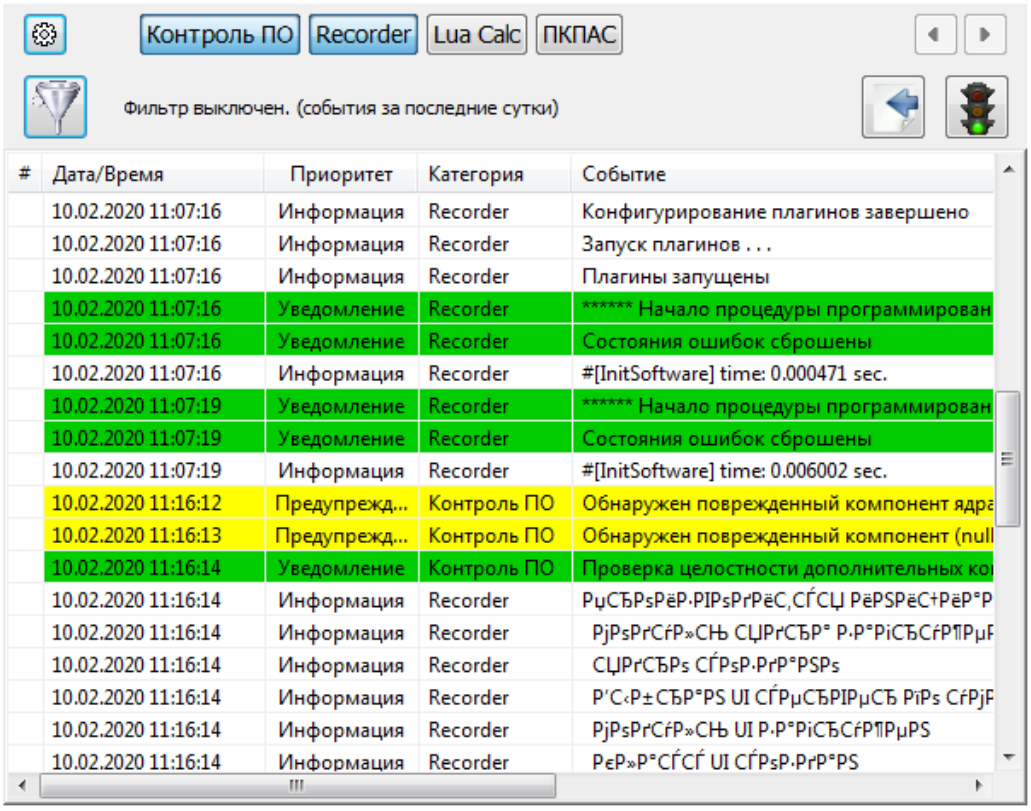


Рисунок 4.31 – Журнал событий

Нажатием  вызывается окно для добавления и выбора категорий сообщений – рисунок 4.32.

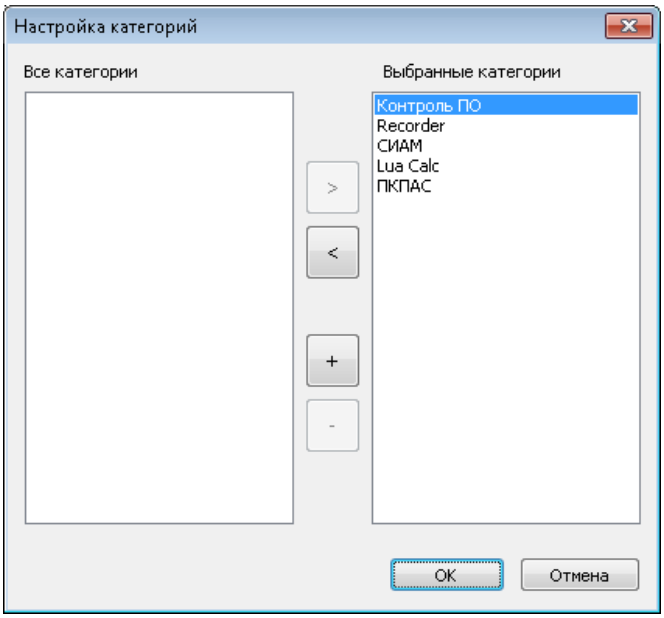


Рисунок 4.32 – Окно для добавления и выбора категорий сообщений

Нажатием  (см. рисунок 4.29) вызывается окно для выбора и включения фильтрации по датам, приоритету, категории и введённому тексту – рисунок 4.33.

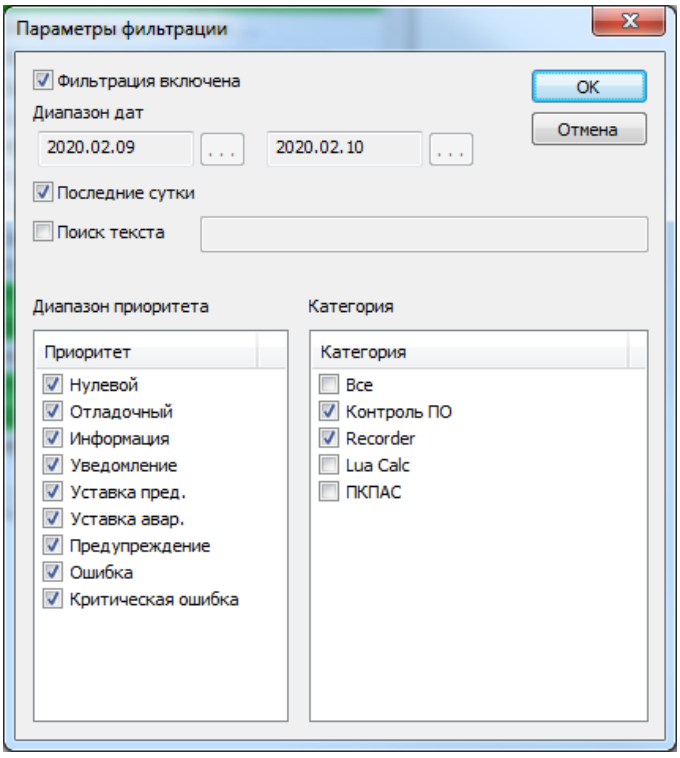


Рисунок 4.33 – Окно выбора и включения фильтрации событий

Журнал событий с включённой фильтрацией показан на рисунке 4.34.

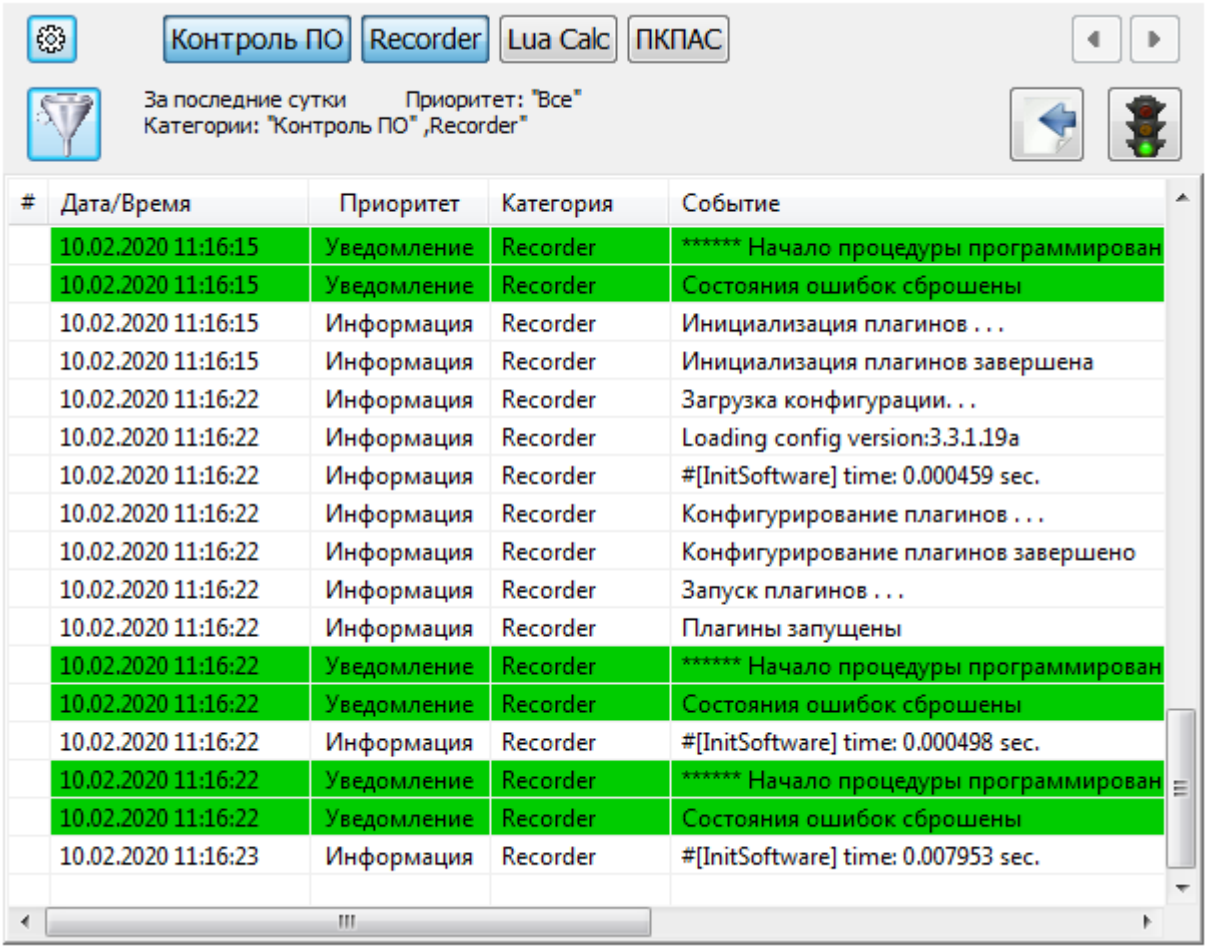


Рисунок 4.34 – Журнал событий с включённой фильтрацией

4.2.13 Добавлением  на мнемосхеме создаётся панель удалённого управления – рисунок 4.35, в том случае, если на стенде необходимо раздельное управление несколькими станциями сбора данных.

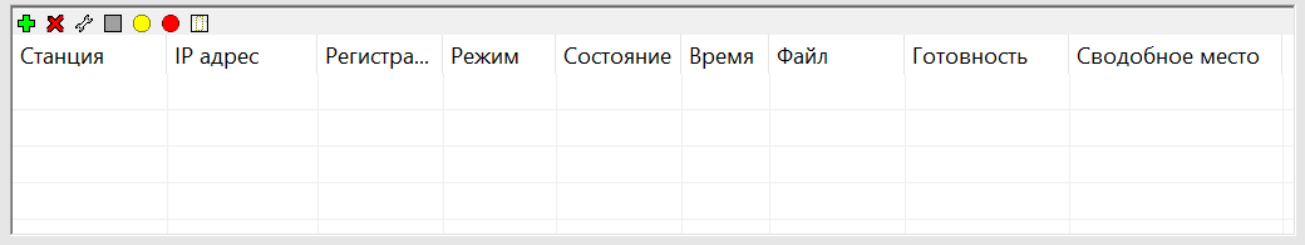


Рисунок 4.35 – Панель удалённого управления

Работа с панелью удалённого управления изложена в “Программное обеспечение удалённого управления и базы данных испытаний. Руководство оператора БЛИЖ.409801.100.130-04 34”.

4.2.14 Добавлением  на мнемосхеме создаётся таблица аварийных ситуаций с выпадающей панелью “Настройка” (при нажатии правой кнопки мыши и вызове “Свойства”) – рисунок 4.36.

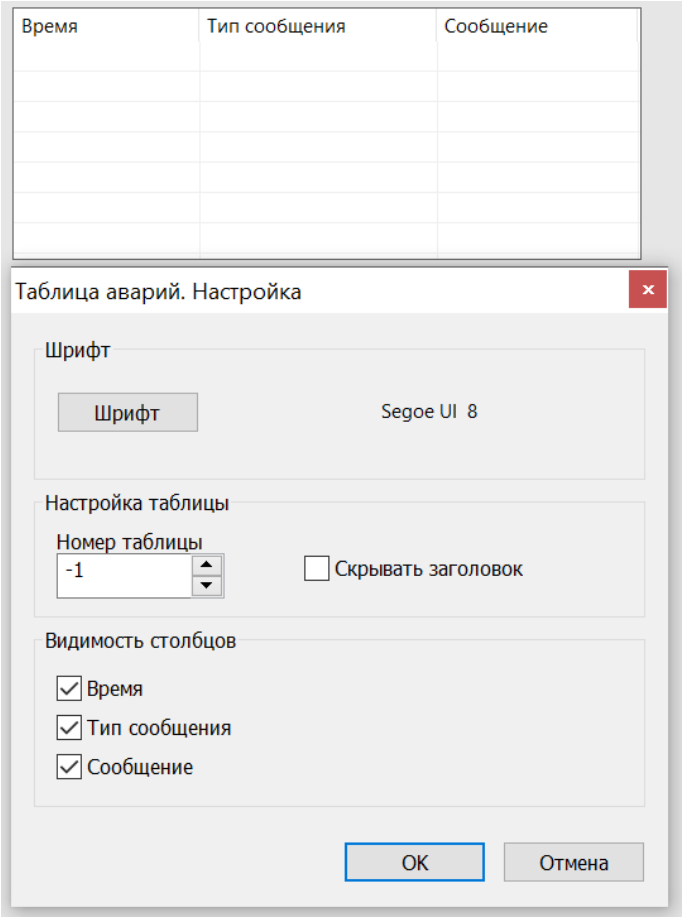


Рисунок 4.36 – Таблица аварийных ситуаций с выпадающей панелью“Настройка”

Ссылки

1 Руководство программиста по разработке расчетных скриптов для Recorder на языке Lua.

2 Lua (язык программирования). Материал из Национальной библиотеки им. Н. Э. Баумана

3 Lua 5.3 Руководство Lua. Roberto Ierusalimschy, Luiz Henrique de Figueiredo, Waldemar Celes. Перевод.