WEBSOCKET SCADA CLIENT

(HMI-V3)

руководство по эксплуатации

2019 - 2024

Оглавление

1. ОБЩИЕ СВЕДЕНИЯ

Реализация:

a) среда разработки
   - любой текстовый редактор

b) язык программирования
   - JavaScript

c) библиотеки / программные платформы
   - jQuery v1.7.2
   - jQuery Mobile 1.4.5

Исполнение:

a) графическое, web-ориентированное
b) web-браузер (сторона клиента)

Поддержка:

a) ОС
   - любая

b) протоколы физического уровня передачи данных
   - Ethernet

c) протоколы прикладного уровня передачи данных
   - WebSocket TCP

d) структуры данных
   - JSON

Связь с сервером сбора данных (Регистратор):

a) Ethernet / WebSocket
b) структура передаваемых данных - JSON
c) клиент подключается и переходит в режим получения данных
d) данные клиенту передаются по готовности (после опроса устройств)
e) клиент может отправлять серверу задание на получение конкретных данных (не всех)
f) клиент может отправлять серверу данные для целевых устройств

Связь с устройствами:

a) через сервер сбора данных (Регистратор)

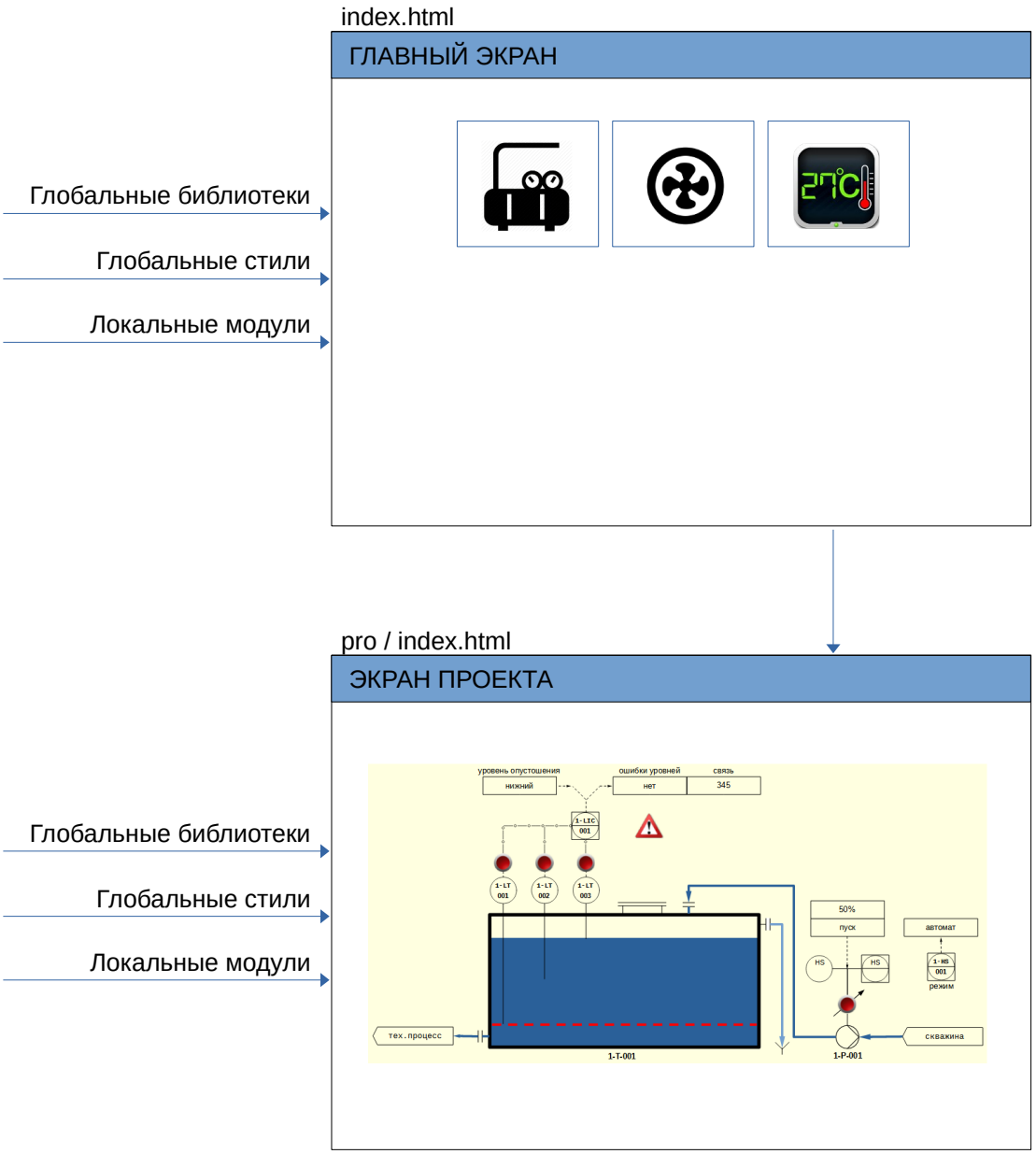
Архивирование данных:

a) через сервер сбора данных (Регистратор)

Настройки:

a) HTML, JavaScript, CSS

2. АРХИТЕКТУРА



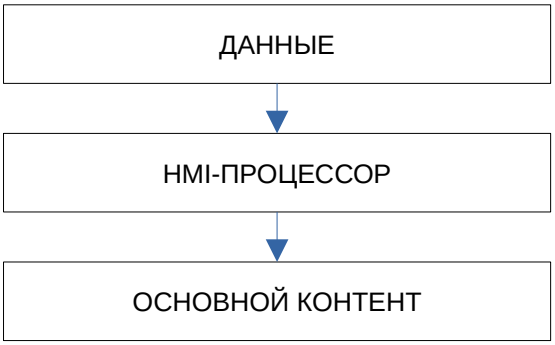
Главный экран:

- список экранов проектов

Экран проекта:

- панель навигации (слева)
  - перемещение между экранами
- основной контент (центр)
  - таблицы
  - мнемосхемы
  - графики
- панель настроек (справа)
  - связь с сервером SCADA и пр.

3. МОДЕЛЬ ДАННЫХ



+ локальные ресурсы (res.js)
+ локальные стилевые классы, изображения

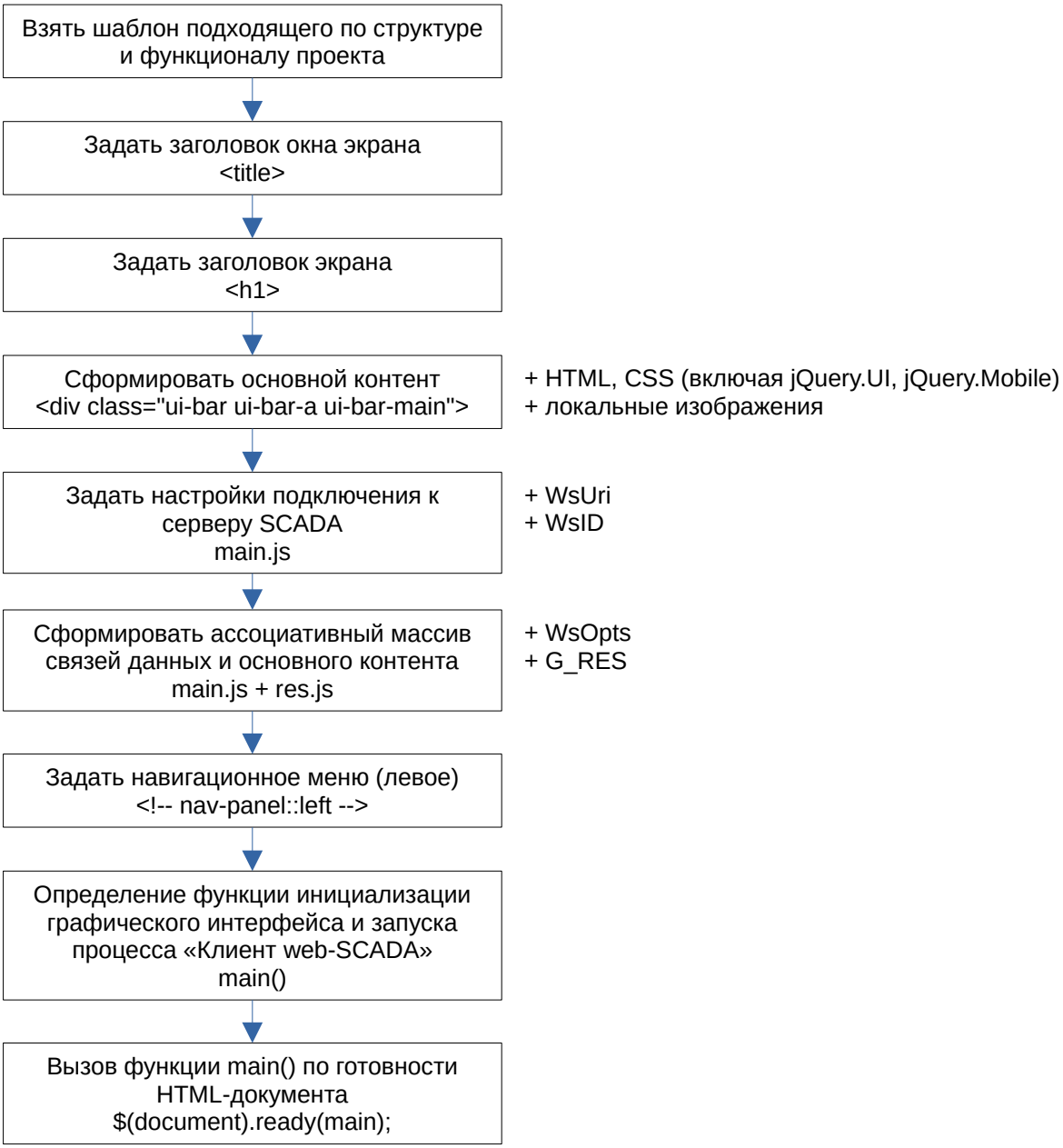
ДАННЫЕ — ассоциативный массив данных от сервера SCADA (см. в РЭ WebSocket Logger).

HMI-ПРОЦЕССОР — обработка данных от сервера SCADA с помощью специальных функций-модификаторов, ассоциация данных с локальными ресурсами и компонентами основного контента. Настройки процессора определяются специальным ассоциативным массивом (WsOpts). Исходный код процессора находится в файле «mod/hmi-v3.js».

ОСНОВНОЙ КОНТЕНТ – HTML-компоненты графического интерфейса, которые будут модифицированы с помощью обработанных данных от сервера SCADA.

4. PRO / INDEX.HTML

4.1 Схема формирования экрана проекта



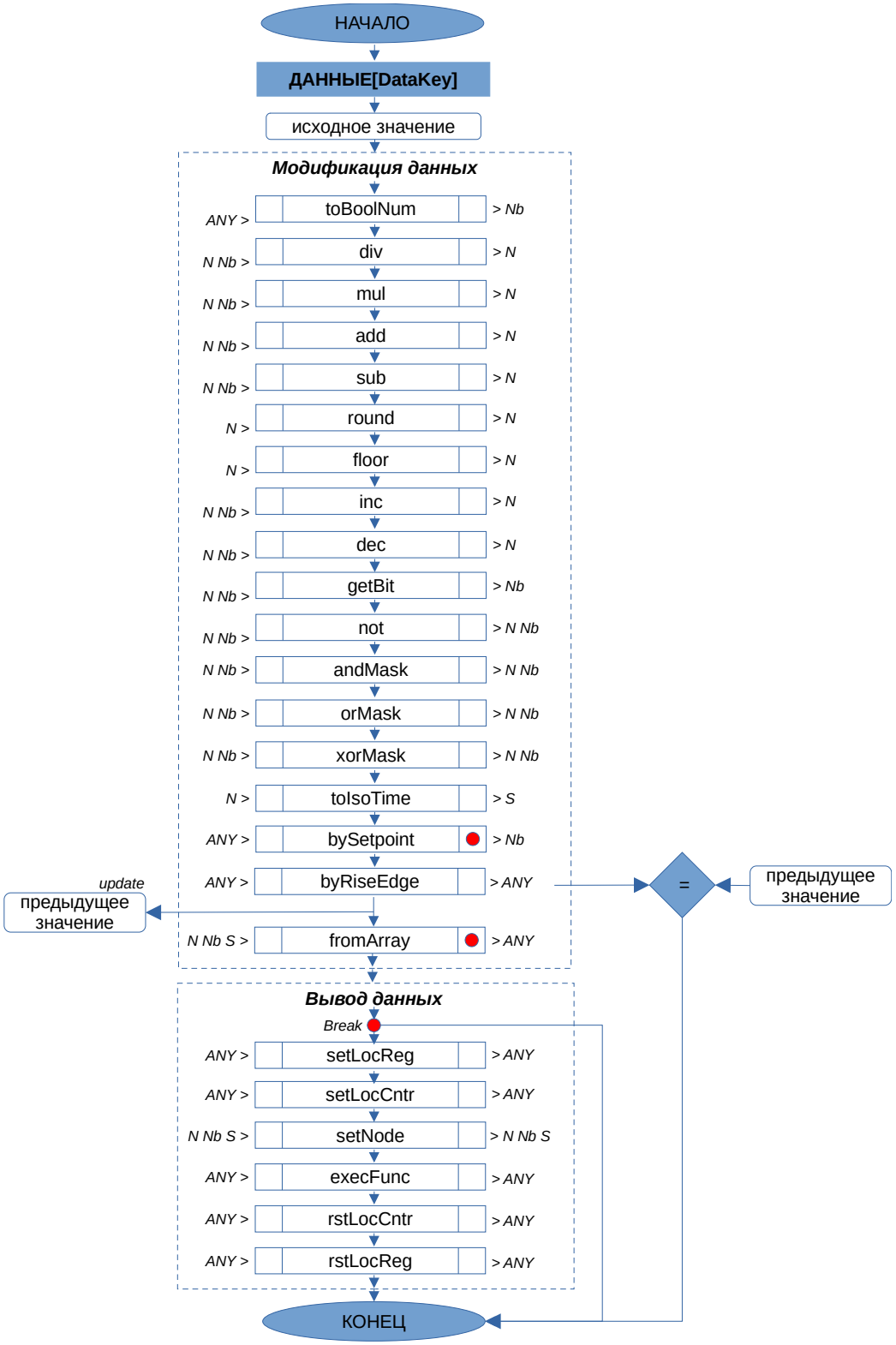
5. PRO / MAIN.JS

5.1 Ассоциативный массив WsOpts

Массив WsOpts определяет поведение HMI-процессора для соответствующего значения, полученного от сервера SCADA:

- извлечение исходного значения из пакета данных
- модификация исходного значения
- управления потоком данных (значений)

Схема одного цикла работы HMI-процессора:



↓ - последовательность выполнения

X > - значение (исходное) на входе метода
> X - значение (модифицированное) на выходе метода
X - тип значения:
- N – тип «number»
- Nb – тип «bool» (0 или 1)
- S – тип «string»
- ANY – любой тип

Исходное значение, извлеченное из массива данных от сервера SCADA, передается в поток обработки:
- изменение, проходя через один или несколько методов-модификаторов
  (при наличии соответствующего набора аргументов)
- вывод на экран клиента (в графический интерфейс)
  (при наличии соответствующего набора аргументов)
- вывод в локальный регистр (для последующей обработки
  (при наличии соответствующего набора аргументов)
- вывод в пользовательскую функцию (для последующей обработки)
  (при наличии соответствующего набора аргументов)

Для одного исходного значения (в одной строке настроек) можно задать несколько методов-модификаторов подряд. В данном случае, методы-модификаторы будут вызываться в порядке, соответствующем схеме работы цикла HMI-процессора (см. выше).

Аналогично, для одного исходного значения (в одной строке настроек) можно задать несколько методов вывода.

Порядок задания настроек для одного значения — не важен (методы будут выполняться в соответствии с циклограммой работы HMI-процессора).

Локальные регистры хранятся в специальном ассоциативном массиве (изначально пустом), который формируется модифицированными исходными значениями по заданным индивидуальным ключам. После обработки всех данных от сервера SCADA, запускается в обработку массив локальных регистров. Соответственно, в контексте WsOpts можно также задать настройки для обработки локальных регистров.

Определение массива:

```
WsOpts = { Key: { Список настроек },
           …
         };
```

где, Key – уникальный ключ к списку настроек
Список настроек — ассоциативный массив настроек поведения HMI-процессора

Пример:

```
WsOpts = { K1Temp:  { DataKey:"K1T", setNode:"K1TNode" },
           K1Press: { DataKey:"K1P", div:10, round:1, setNode:"K1PNode" },
           K1State: { DataKey:"K1Stat", toBoolNum:true, setNode:"K1StatNode",
                      NodeClass:"BitLamp" },
       K1ErrBlink: { DataKey:"K1Err", toBoolNum:true, setNode:"K1ErrNode",
                      NodeClass:"BitLampBlink" },
           …
         };
```

5.1.1 DataKey

Аргумент DataKey содержит имя ключа, по которому извлекается исходное значение из массива данных, полученных от сервера SCADA. Полученное значение далее передается в поток обработки - в блок модификаторов.

Формат:

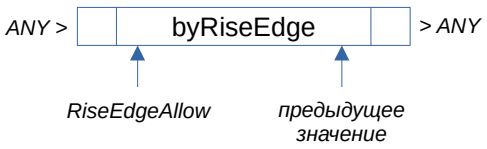
```
Key: { DataKey:”KeyToData”, ... }
```

Результат:

```
Value = DATA["Networks"][i]["Devices"][j][DataKey];
```

где, Value – исходное значение
DATA – массив данных от сервера SCADA (см. в РЭ WebSocket Logger)
i – индекс к списку сетей
j – индекс к списку устройств

5.1.2 byRiseEdge



Метод управления потоком данных:
блокировка повторяемости исходного значения.

Формат:

```
Key: { DataKey:”KeyToData”, byRiseEdge:RiseEdgeAllow, ... }
```

Вход метода:

ANY >	-	исходное значение	[ANY]
`RiseEdgeAllow`	-	разрешение на выполнение метода = 1 или true	[NUMBER || BOOL]
`Предыдущее значение`		передается в метод автоматически (приравнивается исходному значению по окончании потока обработки)	[ANY]

Выход метода:

> ANY	-	исходное значение	[ANY]

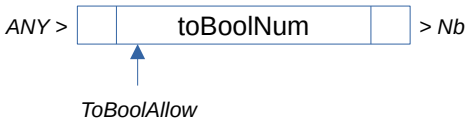
Метод не выполняется, если входные значения не соответствуют требуемому типу.
Метод не модифицирует исходное значение.

Если исходное значение не изменилось (после предыдущего цикла обработки), то поток обработки данного значения завершается (HMI-процессор переходит к обработке следующего значения или, если обработы все данные, то переходит в режим ожидания новых данных от сервера SCADA). Иначе — поток обработки не завершается и исходное значение передается дальше (к следующим методам вывода).

Пример:

```
K1Stat: { DataKey:”K1Stat”, byRiseEdge:true, setNode:”K1Stat” }
---
Предыдущее значение := 999
                ANY > := 1000
K1Stat.setText(1000)
                > ANY := 1000


K1Stat: { DataKey:”K1Stat”, byRiseEdge:true, setNode:”K1Stat” }
---
Предыдущее значение := 1000
                ANY > := 1000
Конец цикла обработки
```

5.1.3 toBoolNum



Метод-модификатор:
преобразование исходного значение в число логической нотации (0 или 1).
= 0 - если исходное значение: 0 [NUMBER], «» [STRING]
= 1 - иначе

Формат:

```
Key: { DataKey:”KeyToData”, toBoolNum:ToBoolAllow, ... }
```

Вход метода:

ANY >	-	исходное значение	[ANY]
ToBoolAllow	-	разрешение на выполнение метода = 1 или true	[NUMBER || BOOL]

Выход метода:

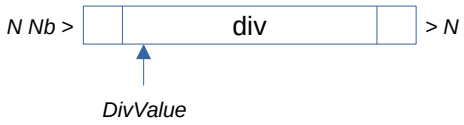
> Nb	-	toBoolNum(исходное значение) == true ? 1 : 0	[NUMBER AS BOOL]

Метод не выполняется, если входные значения не соответствуют требуемому типу — исходное значение не модифицируется и передается далее в поток обработки.

Пример:

```
K1Stat: { DataKey:”K1Stat”, toBoolNum:true, setNode:”K1Stat” }
---
ANY > := 13
  > Nb := 1

K1Stat: { DataKey:”K1Stat”, toBoolNum:true, setNode:”K1Stat” }
---
 ANY > := 0
   > Nb := 0
```

5.1.4 div



Метод-модификатор:
деление исходного значения на число (делитель).

Формат:

```
Key: { DataKey:”KeyToData”, div:DivValue, ... }
```

Вход метода:

N >	-	исходное значение	[NUMBER] [NUMBER AS BOOL]
DivValue	-	Делитель	[NUMBER]

Выход метода:

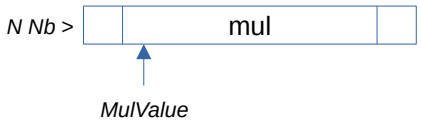
> N	-	(исходное значение) / (делитель)	[NUMBER]

Метод не выполняется, если входные значения не соответствуют требуемому типу — исходное значение не модифицируется и передается далее в поток обработки.

Пример:

```
K1Temp: { DataKey:”K1Temp”, div:100, setNode:”K1Temp” }
---
N > := 3350
> N := 33.50
```

5.1.5 mul



Метод-модификатор:
умножение исходного значения на число (множитель).

Формат:

```
Key: { DataKey:”KeyToData”, mul:MulValue, ... }
```

Вход метода:

N >	-	исходное значение	[NUMBER] [NUMBER AS BOOL]
MulValue	-	множитель	[NUMBER]

Выход метода:

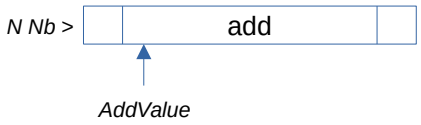
> N	-	(исходное значение) * (множитель)	[NUMBER]

Метод не выполняется, если входные значения не соответствуют требуемому типу — исходное значение не модифицируется и передается далее в поток обработки.

Пример:

```
K1Temp: { DataKey:”K1Temp”, mul:10, setNode:”K1Temp” }
---
N > := 33.50
> N := 335.0
```

5.1.6 add



Метод-модификатор:
прибавление числа (слагаемое) к исходному значению.

Формат:

```
Key: { DataKey:”KeyToData”, add:AddValue, ... }
```

Вход метода:

N >	-	исходное значение	[NUMBER] [NUMBER AS BOOL]
AddValue	-	слагаемое	[NUMBER]

Выход метода:

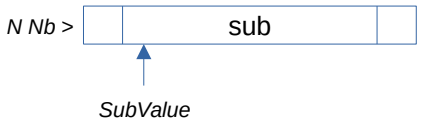
> N	-	(исходное значение) + (слагаемое)	[NUMBER]

Метод не выполняется, если входные значения не соответствуют требуемому типу — исходное значение не модифицируется и передается далее в поток обработки.

Пример:

```
K1Temp: { DataKey:”K1Temp”, add:10, setNode:”K1Temp” }
---
N > := 33.5
> N := 43.5
```

5.1.7 sub



Метод-модификатор:
вычитание числа (вычитаемое) из исходного значения.

Формат:

```
Key: { DataKey:”KeyToData”, sub:SubValue, ... }
```

Вход метода:

N >	-	исходное значение	[NUMBER] [NUMBER AS BOOL]
SubValue	-	вычитаемое	[NUMBER]

Выход метода:

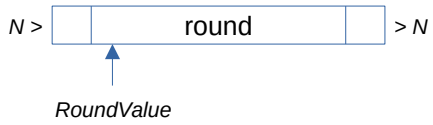
> N	-	(исходное значение) - (вычитаемое)	[NUMBER]

Метод не выполняется, если входные значения не соответствуют требуемому типу — исходное значение не модифицируется и передается далее в поток обработки.

Пример:

```
K1Temp: { DataKey:”K1Temp”, sub:10, setNode:”K1Temp” }
---
N > := 33.5
> N := 23.5
```

5.1.8 round



Метод-модификатор:
округление исходного значения до определенного числа знаков в дробной части.

Формат:

```
Key: { DataKey:”KeyToData”, round:RoundValue, ... }
```

Вход метода:

N >	-	исходное значение	[NUMBER]
RoundValue	-	количество знаков в дробной части	[NUMBER]

Выход метода:

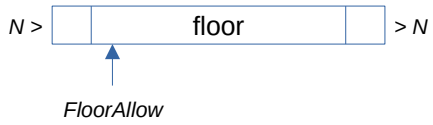
> N	-	(исходное значение).toFixed(RoundValue)	[NUMBER]

Метод не выполняется, если входные значения не соответствуют требуемому типу — исходное значение не модифицируется и передается далее в поток обработки.

Пример:

```
K1Temp: { DataKey:”K1Temp”, round:2, setNode:”K1Temp” }
---
N > := 33.56789
> N := 33.56
```

5.1.9 floor



Метод-модификатор:
приведение исходного значения к ближайшему целому (с округлением в меньшую сторону).

Формат:

```
Key: { DataKey:”KeyToData”, floor:FloorAllow, ... }
```

Вход метода:

N >	-	исходное значение	[NUMBER]
FloorAllow	-	разрешение на выполнение метода = 1 или true	[NUMBER || BOOL]

Выход метода:

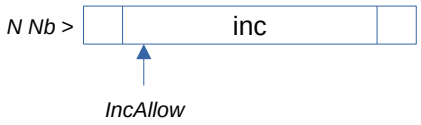
> N	-	Math.floor(исходное значение)	[NUMBER]

Метод не выполняется, если входные значения не соответствуют требуемому типу или не задано разрешение — исходное значение не модифицируется и передается далее в поток обработки.

Пример:

```
K1Temp: { DataKey:”K1Temp”, floor:true, setNode:”K1Temp” }
---
N > := 33.56789
> N := 33

K1Temp: { DataKey:”K1Temp”, floor:0, setNode:”K1Temp” }
---
N > := 33.56789
> N := 33.56789
```

5.1.10 inc



Метод-модификатор:
увеличение исходного значения на 1 (инкремент).

Формат:

```
Key: { DataKey:”KeyToData”, inc:IncAllow, ... }
```

Вход метода:

N >	-	исходное значение	[NUMBER] [NUMBER AS BOOL]
IncAllow	-	разрешение на выполнение метода = 1 или true	[NUMBER || BOOL]

Выход метода:

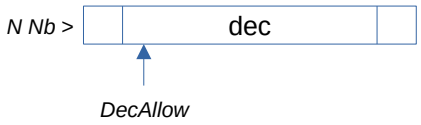
> N	-	(исходное значение)+1	[NUMBER]

Метод не выполняется, если входные значения не соответствуют требуемому типу или не задано разрешение — исходное значение не модифицируется и передается далее в поток обработки.

Пример:

```
K1Temp: { DataKey:”K1Temp”, inc:true, setNode:”K1Temp” }
---
N > := 33
> N := 34

K1Temp: { DataKey:”K1Temp”, inc:0, setNode:”K1Temp” }
---
N > := 33
> N := 33
```

5.1.11 dec



Метод-модификатор:
уменьшение исходного значения на 1 (декремент).

Формат:

```
Key: { DataKey:”KeyToData”, dec:DecAllow, ... }
```

Вход метода:

N >	-	исходное значение	[NUMBER] [NUMBER AS BOOL]
DecAllow	-	разрешение на выполнение метода = 1 или true	[NUMBER || BOOL]

Выход метода:

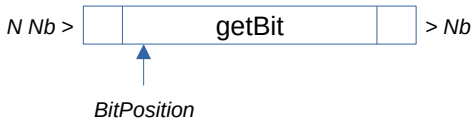
> N	-	(исходное значение)-1	[NUMBER]

Метод не выполняется, если входные значения не соответствуют требуемому типу или не задано разрешение — исходное значение не модифицируется и передается далее в поток обработки.

Пример:

```
K1Temp: { DataKey:”K1Temp”, dec:true, setNode:”K1Temp” }
---
N > := 33
> N := 32

K1Temp: { DataKey:”K1Temp”, dec:0, setNode:”K1Temp” }
---
N > := 33
> N := 33
```

5.1.12 getBit



Метод-модификатор:
извлечение определенного бита из исходного значения.

Формат:

```
Key: { DataKey:”KeyToData”, getBit:BitPosition, ... }
```

Вход метода:

N >	-	исходное значение	[NUMBER] [NUMBER AS BOOL]
BitPosition	-	позиция извлекаемого бита = 0 …	[NUMBER]

Выход метода:

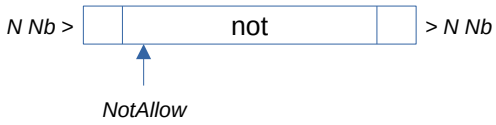
> N	-	getBit(исходное значение, BitPosition)	[NUMBER AS BOOL]

Метод не выполняется, если входные значения не соответствуют требуемому типу — исходное значение не модифицируется и передается далее в поток обработки.

Пример:

```
K1Stat0: { DataKey:”K1States”, getBit:0, setNode:”K1Stat0” }
---
N > := 15
> N := 1
```

5.1.13 not



Метод-модификатор:
поразрядное НЕ для исходного значения.

Формат:

```
Key: { DataKey:”KeyToData”, not:NotAllow, ... }
```

Вход метода:

N >	-	исходное значение	[NUMBER] [NUMBER AS BOOL]
NotAllow	-	разрешение на выполнение метода = 1 или true	[NUMBER || BOOL]

Выход метода:

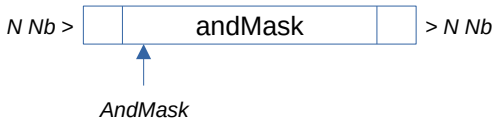
> N Nb	-	~(исходное значение)	[NUMBER] [NUMBER AS BOOL]

Метод не выполняется, если входные значения не соответствуют требуемому типу или нет разрешения — исходное значение не модифицируется и передается далее в поток обработки.

Пример:

```
K1StatInv: { DataKey:”K1Stat”, not:true, setNode:”K1StatInv” }
---
N Nb > := 15
> N Nb := -16
```

5.1.14 andMask



Метод-модификатор:
поразрядное И для исходного значения и маски.

Формат:

```
Key: { DataKey:”KeyToData”, andMask:AndMask, ... }
```

Вход метода:

N >	-	исходное значение	[NUMBER] [NUMBER AS BOOL]
AndMask	-	маска	[NUMBER]

Выход метода:

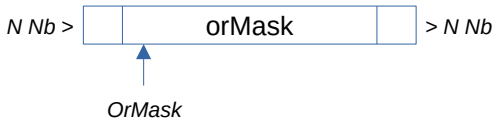
> N	-	(исходное значение) & AndMask	[NUMBER] [NUMBER AS BOOL]

Метод не выполняется, если входные значения не соответствуют требуемому типу — исходное значение не модифицируется и передается далее в поток обработки.

Пример:

```
K1Stat: { DataKey:”K1Stat”, andMask:7, setNode:”K1Stat” }
---
N > := 13
> N := 5
```

5.1.15 orMask



Метод-модификатор:
поразрядное ИЛИ для исходного значения и маски.

Формат:

```
Key: { DataKey:”KeyToData”, orMask:OrMask, ... }
```

Вход метода:

N >	-	исходное значение	[NUMBER] [NUMBER AS BOOL]
OrMask	-	маска	[NUMBER]

Выход метода:

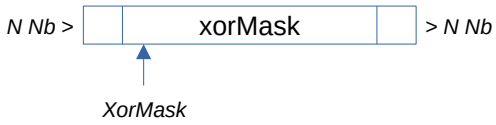
> N	-	(исходное значение) | OrMask	[NUMBER] [NUMBER AS BOOL]

Метод не выполняется, если входные значения не соответствуют требуемому типу — исходное значение не модифицируется и передается далее в поток обработки.

Пример:

```
K1Stat: { DataKey:”K1Stat”, orMask:7, setNode:”K1Stat” }
---
N > := 13
> N := 15
```

5.1.16 xorMask



Метод-модификатор:
поразрядное ИСКЛЮЧАЮЩЕЕ ИЛИ для исходного значения и маски.

Формат:

```
Key: { DataKey:”KeyToData”, xorMask:XorMask, ... }
```

Вход метода:

N >	-	исходное значение	[NUMBER] [NUMBER AS BOOL]
XorMask	-	маска	[NUMBER]

Выход метода:

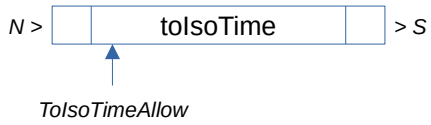
> N	-	(исходное значение) ^ XorMask	[NUMBER] [NUMBER AS BOOL]

Метод не выполняется, если входные значения не соответствуют требуемому типу — исходное значение не модифицируется и передается далее в поток обработки.

Пример:

```
K1Stat: { DataKey:”K1Stat”, xorMask:7, setNode:”K1Stat” }
---
N > := 13
> N := 10
```

5.1.17 toIsoTime



Метод-модификатор:
преобразование исходного числа, заданного в виде метки времени Unix (Unix timestamp – количество секунд с 1970 года), в строку формата ISO (“YYYY-MM-DD HH:MM:SS”).

Формат:

```
Key: { DataKey:”KeyToData”, toIsoTime:ToIsoTimeAllow, ... }
```

Вход метода:

N >	-	исходное значение	[NUMBER]
ToIsoTimeAllow	-	разрешение на выполнение метода = 1 или true	[NUMBER || BOOL]

Выход метода:

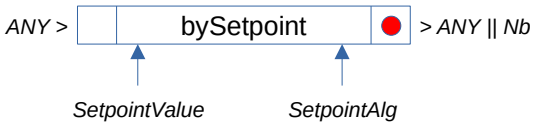
> S	-	toIsoTime(исходное значение)	[STRING]

Метод не выполняется, если входные значения не соответствуют требуемому типу или нет разрешения — исходное значение не модифицируется и передается далее в поток обработки.

Пример:

```
K1Tm: { DataKey:”K1Tm”, toIsoTime:true, setNode:”K1Tm” }
---
 N > := 1659084516
 > S := “2022-07-29 13:48:36”
```

5.1.18 bySetpoint



Метод-модификатор:
сравнение исходного числа с уставкой по определенному алгоритму.

Формат:

```
Key: { DataKey:"KeyToData", bySetpoint:SetpointValue,
                            SetpointAlg:SetpointAlg,
                            SetpointNoMod:true,
                            Break:true,
                            ... }
```

Вход метода:

ANY >	-	исходное значение	[ANY]
`SetpointValue`	-	значение уставки	[ANY]
`SetpointAlg`	-	алгоритм сравнения: для любых типов: = «equ» - Значение == Уставка = «neq» - Значение != Уставка для числовых типов: = «lss» - Значение < Уставка = «leq» - Значение <= Уставка = «gtr» - Значение > Уставка = «geq» - Значение >= Уставка	[STRING]
`SetpointNoMod` *`(необязательный)`*	-	*сравнение без модификации исходного числа* *= 1 или true*	*[NUMBER || BOOL]*
`Break` *`(необязательный)`*	-	*разрешение выхода из потока управления при не совпадении уставки* *= 1 или true*	*[NUMBER || BOOL]*

Выход метода:

> Nb	-	Alg(исходное значение, SetpointValue) ? 1 : 0 ИЛИ немодифицированное исходное значение (если SetpointNoMod:= true)	[NUMBER AS BOOL] [ANY]

Метод не выполняется, если входные значения не соответствуют требуемому типу — исходное значение не модифицируется и передается далее в поток обработки.

Типы данных сравниваемых значений: исходного значения и уставки — должны быть одинаковыми.

Аргумент Break задается только в том случае, когда при несовпадении уставки необходимо завершить поток обработки для текущего исходного значения. Если такое действие не требуется, то аргумент Break в настройках не задается.

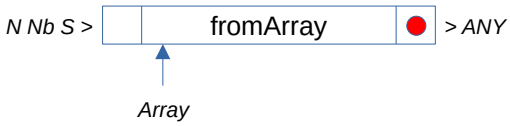
Если задан аргумент SetpointNoMod (= true), то:

- при совпадении исходного значения и уставки возвращается исходное значение,
- при несовпадении — выполняется принудительное завершение потока управления.

Пример:

```
K1Stat: { DataKey:”K1Stat”, bySetpoint:32000,
                            SetpointAlg:”geq”, setNode:”K1Stat” }
---
 ANY > := 13
   > Nb := 0



K1Stat: { DataKey:”K1Stat”, bySetpoint:32000,
                            SetpointAlg:”geq”, setNode:”K1Stat”,
                            Break:true }
---
 ANY > := 13
   > Nb := 0
Конец потока обработки



K1Stat: { DataKey:”K1Stat”, bySetpoint:32000,
                            SetpointAlg:”geq”, setNode:”K1Stat” }
---
 ANY > := 32000
   > Nb := 1



K1Stat: { DataKey:”K1Stat”, bySetpoint:32000,
                            SetpointAlg:”geq”,
                            SetpointNoMod:true,
                             setNode:”K1Stat” }
---
 ANY > := 13
Конец потока обработки



K1Stat: { DataKey:”K1Stat”, bySetpoint:32000,
                            SetpointAlg:”geq”,
                            SetpointNoMod:true,
                             setNode:”K1Stat” }
---
 ANY > := 32000
 > ANY := 32000
```

5.1.19 fromArray



Метод-модификатор:
извлечь значение из массива по ключу (индексу), где ключом является исходное значение.

Формат:

```
Key: { DataKey:”KeyToData”, fromArray:Array,
                            Default:DefaultValue,
                            Break:true, ... }
```

Вход метода:

N Nb S >	-	исходное значение	[NUMBER || NUMBER AS BOOL || STRING]
`Array`	-	указатель на массив или объект (ассоциативный массив)	[ARRAY || OBJECT]
`Default` *`(необязательный)`*	-	*значение по-умолчанию*	[NUMBER || NUMBER AS BOOL || *STRING]*
`Break` *`(необязательный)`*	-	*разрешение выхода из потока управления при отсутствии ключевого значения в массиве = 1 или true*	*[NUMBER || BOOL]*

Выход метода:

> ANY	-	Array[“исходное значение”]	[ANY]

Метод не выполняется, если входные значения не соответствуют требуемому типу — исходное значение не модифицируется и передается далее в поток обработки.

Если массив не имеет заданного ключа (индекса), то, берется значение Default (если задано) или возвращается исходное значение.

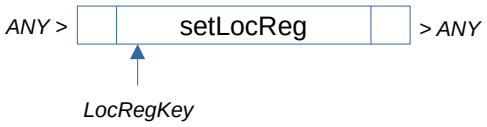
Аргумент Break задается только в том случае, когда при отсутствии ключа необходимо завершить поток обработки для текущего исходного значения. Если такое действие не требуется, то аргумент Break в настройках не задается.

Пример:

```
G_LANG  = “ru”;
G_ARRAY = { ru: {1:”РАБОТА”, 2:”ОСТАНОВ”, 3:”НЕИСПРАВНОСТЬ”} };

K1Stat: { DataKey:”K1Stat”, fromArray:G_ARRAY[G_LANG], setNode:”K1Stat” }
---
N Nb S > := 2
  > ANY := “ОСТАНОВ”
```

```
G_ARRAY = [{a:0, b:1, c:”0-1”}, {a:2, b:3, c:”2-3”}, {a:4, b:5, c:”4-5”}];

K1Stat: { DataKey:”K1Stat”, fromArray:G_ARRAY, setNode:”K1Stat” }
---
N Nb S > := 0
  > ANY := {a:0, b:1, c:”0-1”}
G_ARRAY = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

K1Stat: { DataKey:”K1Stat”, fromArray:G_ARRAY, setNode:”K1Stat” }
---
N Nb S > := -1
  > ANY := -1



G_ARRAY = [{a:0, b:1, c:”0-1”}, {a:2, b:3, c:”2-3”}, {a:4, b:5, c:”4-5”}];

K1Stat: { DataKey:”K1Stat”, fromArray:G_ARRAY, setNode:”K1Stat” }
---
N Nb S > := 0
  > ANY := {a:0, b:1, c:”0-1”}
G_ARRAY = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

K1Stat: { DataKey:”K1Stat”, fromArray:G_ARRAY, Default:”---”, setNode:”K1Stat” }
---
N Nb S > := -1
   > ANY := ”---”



G_ARRAY = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

K1Stat: { DataKey:”K1Stat”, fromArray:G_ARRAY, setNode:”K1Stat”,
                            Break:true }
---
N Nb S > := -1
  > ANY := -1
Конец потока обработки
```

5.1.20 setLocReg



Метод управления потоком данных:
вывод исходного значения в массив локальных регистров.

Формат:

```
Key: { DataKey:”KeyToData”, setLocReg:”LocRegKey”, ... }
```

Вход метода:

ANY >	-	исходное значение	[ANY]
`LocRegKey`	-	имя локального регистра (является ключом к ассоциативному массиву регистров)	[STRING]

Выход метода:

> ANY	-	исходное значение	[ANY]

Метод не выполняется, если входные значения не соответствуют требуемому типу.

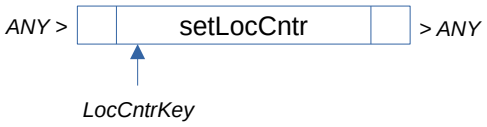
Метод не модифицирует исходное значение — только добавляет его в массив локальных регистров.

В любом случае (успешное выполнение или нет), метод передает (немодифицированное) исходное значение дальше в поток обработки.

Пример:

```
K1Stat: { DataKey:”K1Stat”, add:2, setLocReg:”K1StatBy2”, setNode:”K1Stat” }
---
  ANY > := 1000
 LocReg[”K1StatBy2”] := 1002
  > ANY := 1002
```

5.1.21 setLocCntr



Метод управления потоком данных:
установка (инкремент/декремент) локального регистра-счетчика.

Формат:

```
Key: { DataKey:”KeyToData”, setLocCntr:”LocCntrKey”,
                            LocCntrAlg:”LocCntrAlgValue”,
                            LocCntrMin:0,
                            LocCntrMax:1000 ... }
```

Вход метода:

ANY >	-	исходное значение	[ANY]
`LocCntrKey`	-	имя локального регистра-счетчика (является ключом к ассоциативному массиву регистров)	[STRING]
`LocCntrAlg`	-	алгоритм работы счетчика = «Inc» - только инкремент = «Dec» - только декремент = «IncDec» - инкремент и декремент	[STRING]
`LocCntrMin` *`(необязательный)`*	-	*уставка нижней границы счетчика*	*[NUMBER]*
`LocCntrMax` *`(необязательный)`*	-	*уставка верхней границы счетчика*	*[NUMBER]*

Выход метода:

> ANY	-	исходное значение	[ANY]

Метод не выполняется, если входные значения не соответствуют требуемому типу.
Метод не модифицирует исходное значение.

Счетчик начинает отсчет от 0.

Если LocCntrAlg = «Dec» или «IncDec»
И исходное значение = 0 (number), или «» (string), или false (boolean), или null,
то знаение регистра-счетчика декрементируется (с шагом 1).

Если LocCntrAlg = «Inc» или «IncDec»
И исходное значение > 0 (number), или не пустая строка (string), или true (boolean),
то знаение регистра-счетчика инкрементируется (с шагом 1).

Если задан аргумент LocCntrMin, то счетчик будет декрементировать до этого значения.
Если задан аргумент LocCntrMax, то счетчик будет инкрементировать до этого значения.

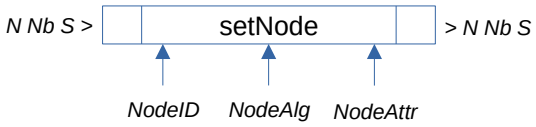
Сброс счетчика осущесвтляется методом rstLocCntr.

В любом случае, немодифицированное исходное значение по завершении работы метода передается далее в поток обработки.

Пример:

```
K1Stat: { DataKey:”K1Stat”, byRiseEdge:true, setLocCntr:”K1StatCntr” }
---
   ANY > := 1000 (previus 999)
  LocReg[”K1StatCntr”] := 1
   > ANY := 1000


K1Stat: { DataKey:”K1Stat”, byRiseEdge:true, setLocCntr:”K1StatCntr” }
---
   ANY > := 1000 (previus 1000)
Конец цикла обработки


K1Stat: { DataKey:”K1Stat”, byRiseEdge:true, setLocCntr:”K1StatCntr” }
---
   ANY > := 1001 (previus 1000)
  LocReg[”K1StatCntr”] := 2
   > ANY := 1001
```

5.1.22 setNode



Метод управления потоком данных:
вывод исходного значения на экран (в HTML-контекст).

Формат:

```
Key: { DataKey:”KeyToData”, setNode:”NodeID”,
                             NodeAlg:”NodeAlg”,
                           NodeAttr:”NodeAttr”, ... }
```

Вход метода:

N Nb S >	-	исходное значение	[NUMBER] [NUMBER AS BOOL] [STRING]
`NodeID`	-	идентификатор целевого узла HTML-контента	[STRING]
`NodeAlg`	-	алгоритм вывода значения: = «Text» - вывести как текст непосредственно в целевой узел (используется по-умолчанию) = «Attr» - вывести в указанный атрибут (NodeAttr) целевого узла = «BitLamp» - скрыть целевой узел, если исходное значение: 0 [NUMBER], «» [STRING] - отобразить целевой узел, если иначе = «BitLampBlink» - скрыть целевой узел, если исходное значение: 0 [NUMBER], «» [STRING] - отобразить целевой узел в режиме периодического (T=1сек) мигания, если иначе	[STRING]
`NodeAttr`	-	имя атрибута целевого узла (если NodeAlg:«Attr») = «title» (заменяется полностью) = «class» (заменяется по значению) = «style» (заменяется полностью) … (заменяется полностью)	[STRING]

Выход метода:

> ANY - исходное значение [ANY]

Метод не выполняется, если входные значения не соответствуют требуемому типу.
Метод не модифицирует исходное значение.

Если NodeAlg не задан, то используется алгоритм «Text».

Принцип работы алгоритма «BitLamp»:

- если целевой узел - изображение, то для него можно задать следующие атрибуты:
  - src - изображение по-умолчанию
  - src0 - изображение, когда исходное значение равно FALSE (пустая строка, или 0, или false)
  - src1 - изображение, когда исходное значение равно TRUE (непустая строка, или > 0, или true)
- иначе — будет использован метод «hide / show» для всего целевого узла.

Алгоритм «BitLampBlink» использует только метод «hide / show».

Пример:

```
K1Stat: { DataKey:”K1Stat”, setNode:”K1Stat” }
---
  ANY > := 1000
K1Stat.setText(1000)
```

```
G_LANG  = “ru”;
G_ARRAY = { ru: {1:”РАБОТА”, 2:”ОСТАНОВ”, 3:”НЕИСПРАВНОСТЬ”} };

K1Stat: { DataKey:”K1Stat”, fromArray:G_ARRAY[G_LANG],
                            setNode:”K1Stat” }
---
    ANY > := “РАБОТА”
K1Stat.setText(“РАБОТА”)
   > ANY := “РАБОТА”
```

```
G_LANG  = “ru”;
G_ARRAY = { ru: {0:”ui-none”, 1:”ui-red”} };

K1Stat: { DataKey:”K1Stat”, toBoolNum:true, fromArray:G_ARRAY[G_LANG],
                            setNode:”K1Stat”,
                            NodeAlg:”Attr”,
                           NodeAttr:”class” }
---
   ANY > := “ui-red”
K1Stat.attr(“class”, “ui-red”)
   > ANY := “ui-red”
```

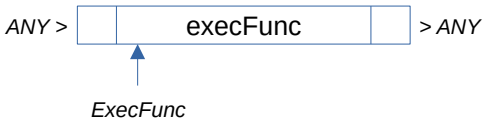
```
K1Stat: { DataKey:”K1Stat”, toBoolNum:true,
                            setNode:”K1Stat”,
                            NodeAlg:”BitLamp” }
```

```
---
```
ANY > := 0
K1Stat.hide()
> ANY := 0

```
K1Stat: { DataKey:”K1Stat”, toBoolNum:true,
                            setNode:”K1Stat”,
                            NodeAlg:”BitLamp” }
---
```
ANY > := 1
K1Stat.show()
> ANY:= 1

```
K1Stat: { DataKey:”K1Stat”, toBoolNum:true,
                            setNode:”K1Stat”,
                            NodeAlg:”BitLampBlink” }
---
```
ANY > := 1
K1Stat.blink(1000) //T=1000ms
> ANY:= 1

5.1.23 execFunc



Метод управления потоком данных:
исполнение пользовательской функции.

Формат:

```
Key: { DataKey:”KeyToData”, execFunc:ExecFunc, ... }
```

Вход метода:

ANY >	-	исходное значение	[ANY]
`ExecFunc`	-	пользовательская функция	[FUNCTION]

Выход метода:

> ANY	-	исходное значение	[ANY]

Метод не выполняется, если входные значения не соответствуют требуемому типу.
Метод не модифицирует исходное значение.

В пользовательскую функцию передаются следующие аргументы:

Options	-	ассоциативный массив настроек для обработки исходного значения	[OBJECT]
Data	-	массив данных от сервера SCADA	[OBJECT]
DataKey	-	уникальный ключ к списку данных для исходного значения	[STRING]
Value	-	исходное значение (модифицированное, включая ассоциацию через fromArray – если было определено)	[ANY]
BeforeValue	-	исходное значение (модифицированное, но до ассоциации через fromArray)	[ANY]

Пример:

```
K1Stat: { DataKey:”K1Stat”, execFunc:myFunc }
---
  ANY > := 1000
myFunc(WsOpts[“K1Stat”], DATA, “K1Stat”, 1000);
  > ANY := 1000
```

5.1.23.1 onFuncWsLog

Реализация пользовательской функции:
вывод исходного значения в журнал событий или в системную консоль браузера.

Формат:

```
Key: { DataKey:”KeyToData”, execFunc:onFuncWsLog,
                            LogTarget:”Target”,
                           LogColor:Color }
```

Формат выводимого сообщения:

TIMESTAMP TARGET MESSAGE

где, TIMESTAMP – строка даты и времени в формате ISO (добавляется автоматически)
TARGET – имя объекта, к которому относится сообщение [STRING]
MESSAGE – сообщение [STRING || NUMBER || BOOL]

Имя объекта (TARGET) необходимо передать через аргумент «LogTarget» списка настроек.
Сообщение (MESSAGE) соответствует исходному значению.

Изменить цвет строки сообщения можно, задав нужное значение через аргумент «LogColor»:

- в виде строковой константы [STRING]
  - “red” — красный
  - “yellow” – желтый
  - “green” - зеленый
  - “blue” – синий
- в виде указателя на ассоциативный массив, где модифицированному до fromArray исходному значению, переданному в метод execFunc, сопоставлен определенный цвет в виде строки

Пример:

```
G_LANG  = “ru”;
G_ARRAY = { ru: {K1:”Компрессор 1”,
                 Stat:{1:”РАБОТА”, 0:”ОСТАНОВ”}
                 }
          };

K1Stat: { DataKey:”K1Stat”, fromArray:G_ARRAY[G_LANG][“Stat”],
                            execFunc:onFuncWsLog,
                            LogTarget:G_ARRAY[G_LANG][“K1”] }
---
DATA[DataKey] := 1
fromArray(G_ARRAY[G_LANG][“Stat”], DATA[DataKey]) := “РАБОТА”
onFuncWsLog( WsOpts[“K1Stat”], DATA, “K1Stat”, “РАБОТА”);
  > ANY := “РАБОТА”
```

2022-08-01 09:35 Компрессор 1 РАБОТА

```
K1Stat: { DataKey:”K1Stat”, fromArray:G_ARRAY[G_LANG][“Stat”],
                            execFunc:onFuncWsLog,
                            LogTarget:G_ARRAY[G_LANG][“K1”],
```

```
                                LogColor:”red” }
```

2022-08-01 09:35 Компрессор 1 РАБОТА

```
K1Stat: { DataKey:”K1Stat”, fromArray:G_ARRAY[G_LANG][“Stat”],
                            execFunc:onFuncWsLog,
                            LogTarget:G_ARRAY[G_LANG][“K1”],
                            LogColor:”yellow” }
```

2022-08-01 09:35 Компрессор 1 РАБОТА

```
K1Stat: { DataKey:”K1Stat”, fromArray:G_ARRAY[G_LANG][“Stat”],
                            execFunc:onFuncWsLog,
                            LogTarget:G_ARRAY[G_LANG][“K1”],
                            LogColor:”green” }
```

2022-08-01 09:35 Компрессор 1 РАБОТА

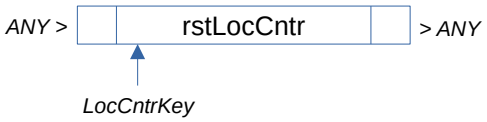
```
K1Stat: { DataKey:”K1Stat”, fromArray:G_ARRAY[G_LANG][“Stat”],
                            execFunc:onFuncWsLog,
                            LogTarget:G_ARRAY[G_LANG][“K1”],
                            LogColor:”blue” }
```

2022-08-01 09:35 Компрессор 1 РАБОТА

```
G_LANG  = “ru”;
G_ARRAY = { ru: {K1:”Компрессор 1”,
                 Stat:{1:”РАБОТА”, 0:”ОСТАНОВ”},
                 StatLogColor:{1:null, 0:”red”}
                 }
           };

K1Stat: { DataKey:”K1Stat”, fromArray:G_ARRAY[G_LANG][“Stat”],
                            execFunc:onFuncWsLog,
                            LogTarget:G_ARRAY[G_LANG][“K1”],
                            LogColor:G_ARRAY[G_LANG][“StatLogColor”] }
---
DATA[DataKey] := 0
fromArray(G_ARRAY[G_LANG][“Stat”], DATA[DataKey]) := “ОСТАНОВ”
onFuncWsLog( WsOpts[“K1Stat”], DATA, “K1Stat”, “ОСТАНОВ”);
  > ANY := “ОСТАНОВ”
```

2022-08-01 09:35 Компрессор 1 ОСТАНОВ

5.1.24 rstLocCntr



Метод управления потоком данных:
сброс (обнуление) локального регистра-счетчика.

Формат:

```
Key: { DataKey:”KeyToData”, rstLocCntr:”LocCntrKey”, ... }
```

Вход метода:

ANY >	-	исходное значение	[ANY]
`LocCntrKey`	-	имя локального регистра-счетчика (является ключом к ассоциативному массиву регистров)	[STRING]

Выход метода:

> ANY	-	исходное значение	[ANY]

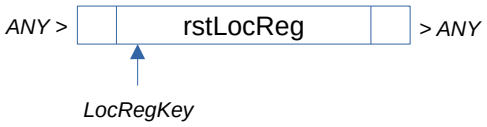
Метод не выполняется, если входные значения не соответствуют требуемому типу.
Метод не модифицирует исходное значение.

Если в массиве регистров нет элемента с ключом LocCntrKey, то ничего не выполняется — немодифицированное исходное значение передается далее в поток обработки.

Пример:

```
K1Stat: { DataKey:”K1Stat”, rstLocReg:”K1StatCntr” }
---
  ANY > := 1000
 LocReg[”K1StatCntr”] := 0
  > ANY := 1000
```

5.1.25 rstLocReg



Метод управления потоком данных:
сброс (обнуление) локального регистра.

Формат:

```
Key: { DataKey:”KeyToData”, rstLocReg:”LocRegKey”, ... }
```

Вход метода:

ANY >	-	исходное значение	[ANY]
`LocRegKey`	-	имя локального регистра (является ключом к ассоциативному массиву регистров)	[STRING]

Выход метода:

> ANY	-	исходное значение	[ANY]

Метод не выполняется, если входные значения не соответствуют требуемому типу.
Метод не модифицирует исходное значение.

Если в массиве регистров нет элемента с ключом LocCntrKey, то ничего не выполняется — немодифицированное исходное значение передается далее в поток обработки.

Пример:

```
K1Stat: { DataKey:”K1Stat”, rstLocReg:”K1StatBy2” }
---
  ANY > := 1002
 LocReg[”K1StatBy2”] := 0
  > ANY := 1002
```