NaftaProcess®

НЕФТЕАВТОМАТИКА

Руководство по работе с пользовательскими сценариями

2025

Содержание

1. Условные обозначения и термины....4
1.1. Условные обозначения....4
1.2. Перечень терминов и сокращений....4

2. Введение....5

3. Операции с пользовательскими сценариями....6
3.1. Создание пользовательского сценария....6
3.2. Переименование пользовательского сценария....11
3.3. Импорт пользовательского сценария....12
3.4. Экспорт пользовательского сценария....14
3.5. Удаление пользовательского сценария....15

4. Использование псевдонимов в пользовательском сценарии....17

5. Библиотека пользовательских сценариев....18
5.1. Авторизация и управление пользователями....18
5.1.1. logIn()....18
5.1.2. logOut()....18
5.2. Отправка сообщений....19
5.2.1. eventSender....19
5.3. Чтение и запись данных в теги....20
5.3.1. readValue(tagname)....21
5.3.2. writeValue(tagname, value)....21
5.4. Чтение и запись данных в свойства визуальных объектов....22
5.4.1. readObjectProperty(id, property)....23
5.4.2. writeObjectProperty(id, property, value)....23
5.4.3. Особенности функции writeObjectProperty....24
5.5. Открытие и закрытие окна в режиме исполнения....25
5.5.1. closeWindow()....25
5.5.2. closeWindow(name, displayNumber)....25
5.5.3. openWindow(name, title, windowParameters, x, y, aliasValues, displayNumber, closeTime, closeWhenCovered, openIfAlreadyOpened)....26
5.5.4. windowSettings()....32
5.6. Окно ввода значений....34
5.6.1. inputWindow(width, height, x, y, title, defaultValue, dataType)....34

5.6.2. inputWindow(width, height, title, defaultValue, dataType)..... 37
5.7. Координаты окна..... 38
5.8. Запуск стороннего приложения..... 38
5.8.1. runCmd(command,arguments)..... 38
5.8.2. runCmdAndWait(command,arguments)..... 39
5.9. Подключение к базе данных..... 40
5.9.1. Настройка resultSetModes..... 42
5.10. Вывод логов..... 44
5.11. Чтение данных Modbus..... 45
5.11.1. readModbusArray(channel, tagName, left, right)..... 45
5.11.1.1. Квитирование сигнализации..... 46
5.12. Квитирование сигнализации..... 47
5.13. Подавление (откладывание) сигнализации..... 47
5.14. Управление звуками сигнализации..... 49
5.14.1. acknowledgeSound()..... 49
5.14.2. beep()..... 49
5.14.3. muteSound(mute)..... 50
5.14.4. playSound(name)..... 50
5.15. Окно просмотра отчетов..... 51
5.16. Закрытие окна режима исполнения..... 51
5.17. Вызов методов OPC сервера..... 52
5.18. Доступ к фильтрам сигнализации..... 53
5.18.1. alarmFilterSettings()..... 53
5.18.2. alarmFilterSettings(filterSource)..... 54

1. Условные обозначения и термины

1.1. Условные обозначения

Внимание:

Помечает информацию, с которой необходимо ознакомиться, чтобы учесть особенности работы какого-либо элемента программного обеспечения.

ОСТОРОЖНО:

Помечает информацию, с которой необходимо ознакомиться, чтобы предотвратить нарушения в работе программного обеспечения либо предотвратить потерю данных.

ОПАСНО:

Помечает информацию, с которой необходимо ознакомиться, чтобы избежать потери контроля над технологическим процессом.

1.2. Перечень терминов и сокращений

Проект

Набор данных, который представляет конфигурацию SCADA.

Значение по умолчанию

Предопределенное значение какого-либо аргумента. Если пользователь не передает в функцию значение для такого аргумента, то будет использоваться значение по умолчанию. Если же пользователь передает значение, то это значение будет использоваться вместо значения по умолчанию.

Функция может использовать несколько аргументов по умолчанию, но, например в функции *function(a, b, c)* предоставить значение для аргумента *c*, не предоставляя при этом аргументы для параметров *a* и *b* — нельзя, "перепрыгнуть" через аргумент невозможно.

2. Введение

Документ "Руководство по работе с пользовательскими сценариями " (далее Руководство) относится к комплекту эксплуатационных документов программного обеспечения.

Руководство содержит следующую информацию:

- описание принципов работы с пользовательскими сценариями;
- особенности использования псевдонимов в пользовательском сценарии;
- библиотеку пользовательских сценариев.

 **Внимание:** Справочная информация доступна:

- из главного меню командой **Помощь** > **Справка**;
- по клавише **“F1”**;
- выбором пункта **Справка** из контекстного меню дерева проекта.

3. Операции с пользовательскими сценариями

В проекте предусмотрены следующие операции с пользовательскими сценариями:

- создание;
- переименование;
- импорт;
- экспорт;
- удаление.

3.1. Создание пользовательского сценария

Для того чтобы создать пользовательский сценарий в проекте, выполните следующие действия:

1. Откройте вкладку **Конфигурация** в дереве проекта.
2. Правой кнопкой мыши выберите вкладку **Пользовательские сценарии**.
3. В открывшемся контекстном меню выберите **Создать пользовательский сценарий**:

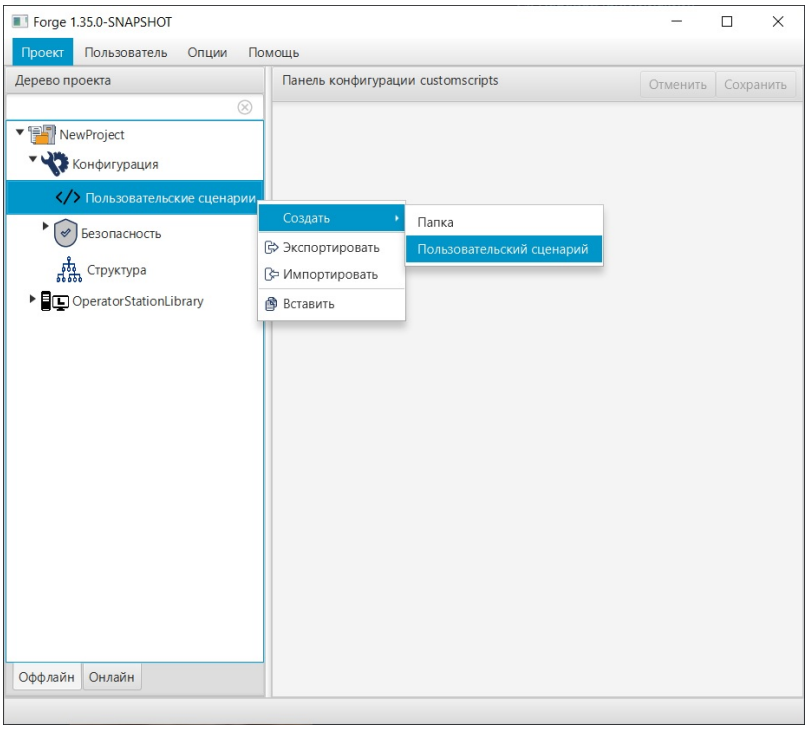

Рисунок 1. Создание пользовательского сценария

Пользовательский сценарий добавится в дерево проекта во вкладку **Пользовательские сценарии**:

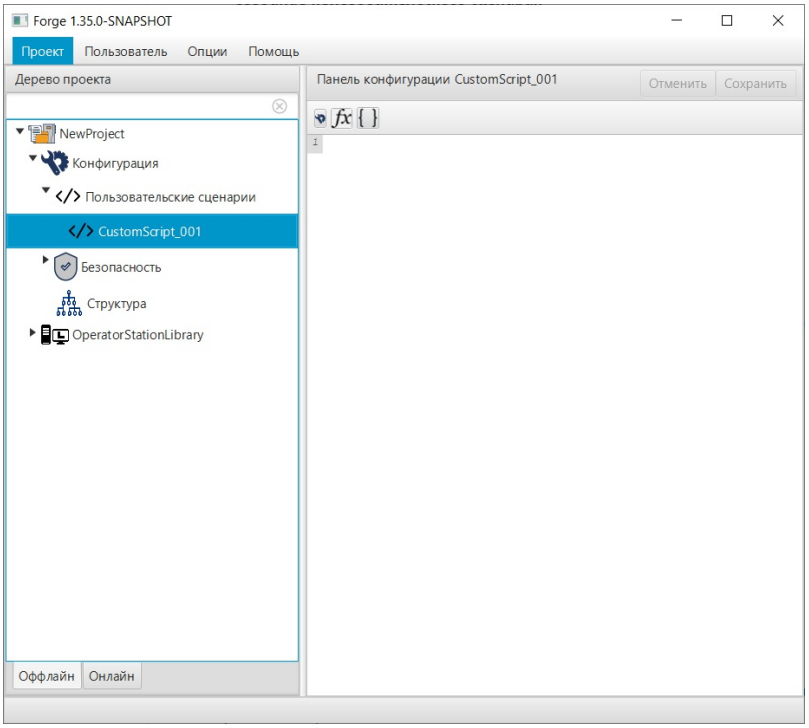

Рисунок 2. Пользовательский сценарий в дереве проекта

4. Нажмите кнопку на панели редактора. В открывшемся окне задайте параметры редактора (шрифт, цвет заливки).

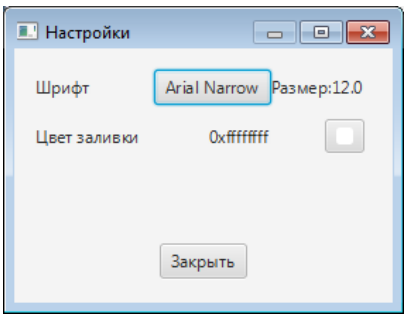

Рисунок 3. Кнопка "Настройки"

5. Введите текст пользовательского сценария в окно конфигурации одним из способов:

- вручную;
- нажмите на панели кнопку fx либо сочетание клавиш “Alt”+“F”. В открывшемся окне выберите функцию. Есть возможность выбрать полную либо сокращенную запись функции. При сокращенной записи в качестве атрибутов для удобства используются символы "a", "b" и т. п.

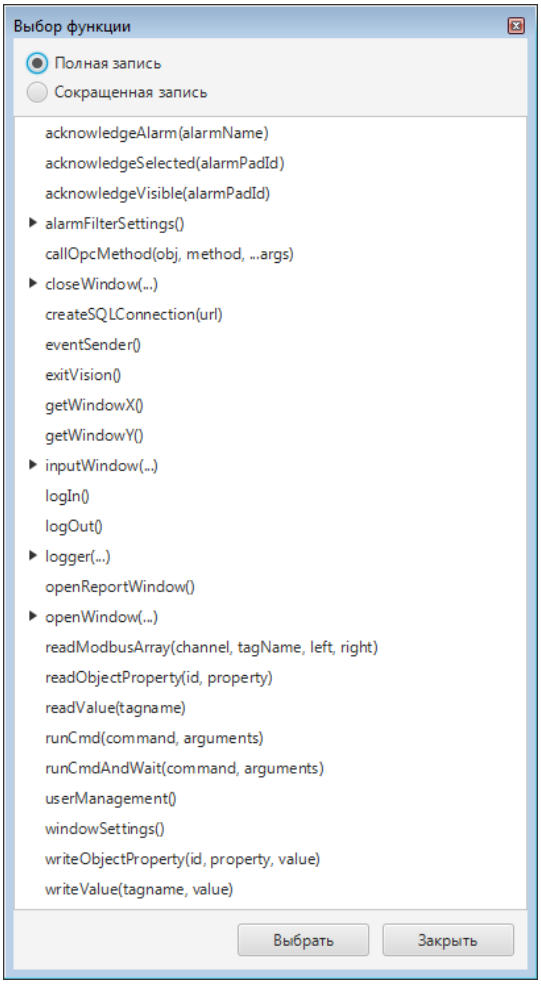

Рисунок 4. Полная запись

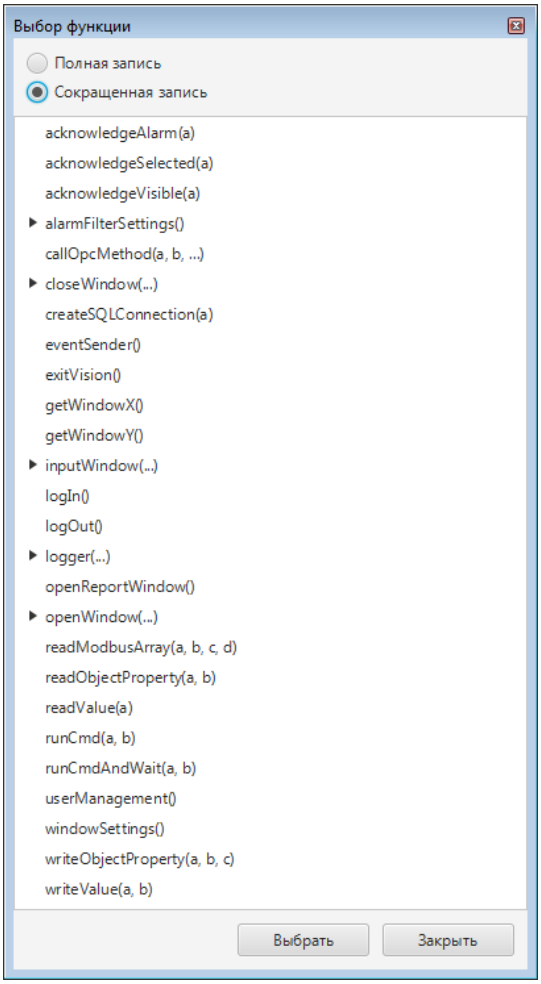

Рисунок 5. Сокращенная запись

Внимание: В подсказке отображается полная запись функции с указанием типа данных атрибута.

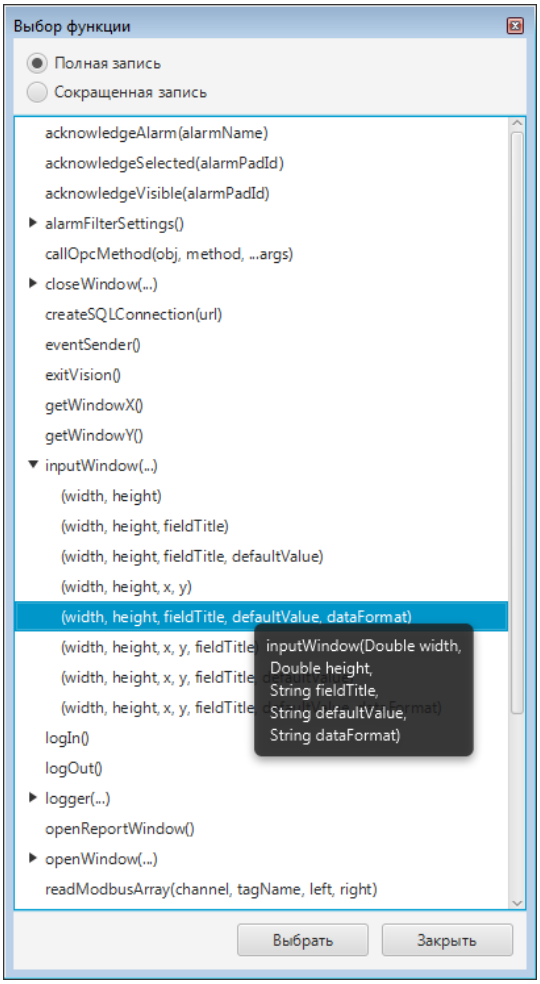

Рисунок 6. Всплывающая подсказка

6. Для использования тега в качестве атрибута функции, нажмите кнопку { } на панели редактора. В окне **Выбор тега** выделите нужный тег и нажмите кнопку **ОК** либо дважды щелкните левой кнопкой мыши по тегу. Имя тега отобразится в выбранном поле.

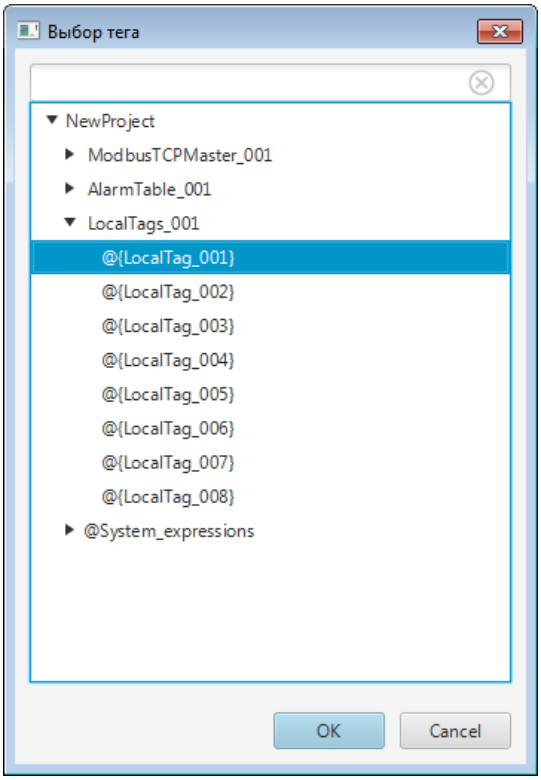

Рисунок 7. Выбор из дерева тегов

7. Сохранение пользовательского сценария не требуется.

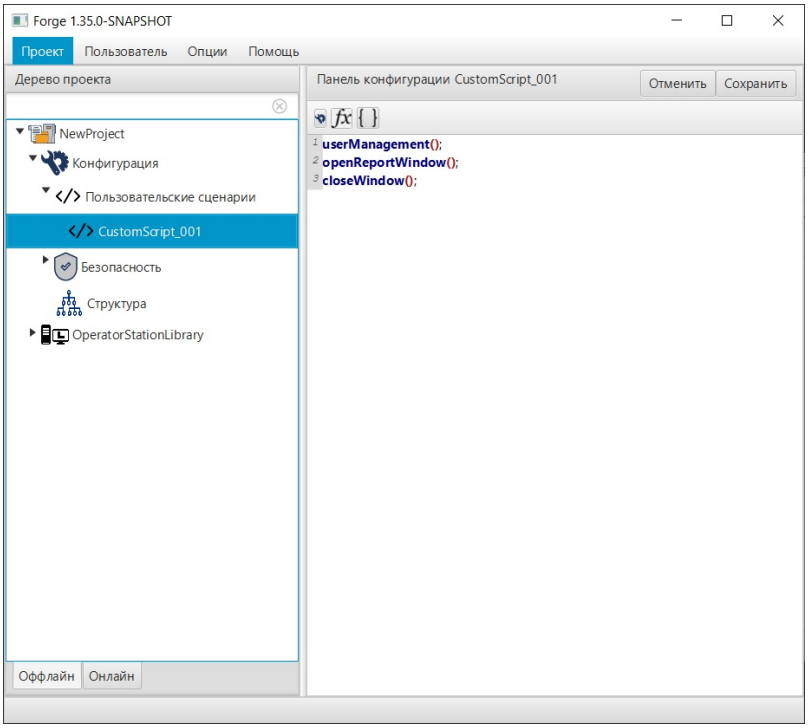

Рисунок 8. Окно конфигурации пользовательского сценария

3.2. Переименование пользовательского сценария

Для того чтобы переименовать пользовательский сценарий, выполните следующие действия:

1. В дереве проекта откройте вкладки **Конфигурация** > **Пользовательские сценарии**.
2. Выберите нужный пользовательский сценарий правой кнопкой мыши.
3. В контекстном меню выберите **Переименовать**.

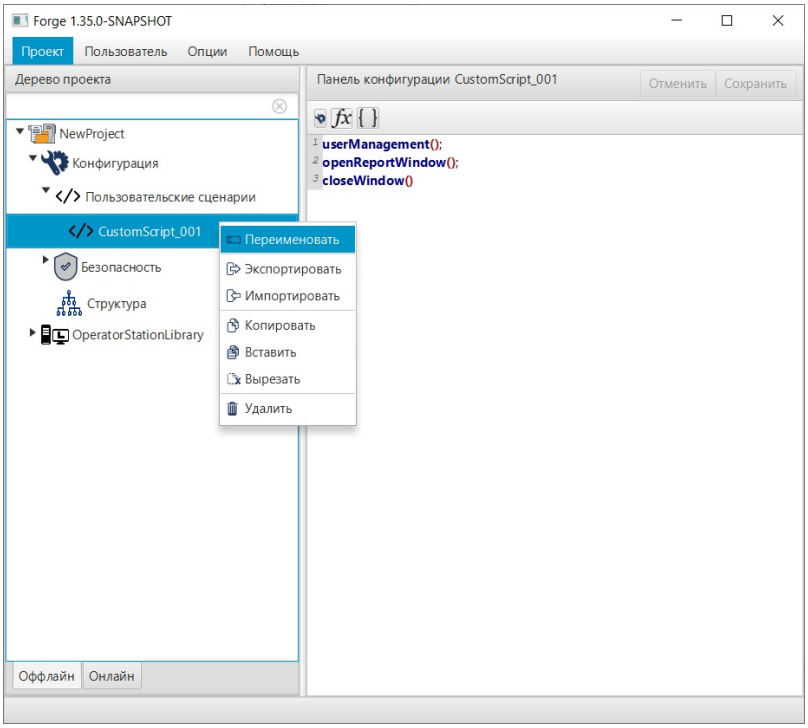

Рисунок 9. Переименование пользовательского сценария

4. В диалоговом окне измените имя пользовательского сценария и нажмите **ОК**:

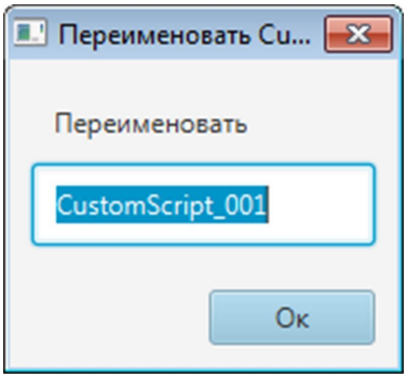

Рисунок 10. Новое имя пользовательского сценария

В дереве проекта сценарий отобразится с новым именем.

3.3. Импорт пользовательского сценария

Для того чтобы импортировать ранее созданный пользовательский сценарий в проект, выполните следующие действия:

1. В дереве проекта правой кнопкой мыши выберите пользовательский сценарий, в который нужно импортировать данные.

> **ОСТОРОЖНО:** Во время операции *Импорт* данные выбранного пользовательского сценария **полностью** заменяются на импортируемые данные.

2. В открывшемся контекстном меню выберите **Импортировать**:

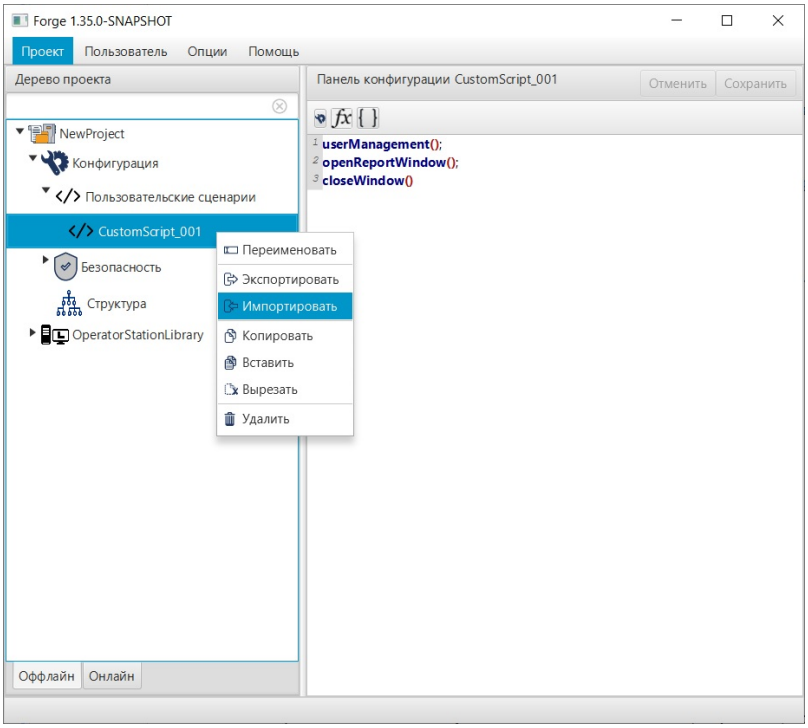

Рисунок 11. Импорт пользовательского сценария

3. В открывшемся окне **Импорт *<имя пользовательского сценария>*** укажите путь расположения файла:

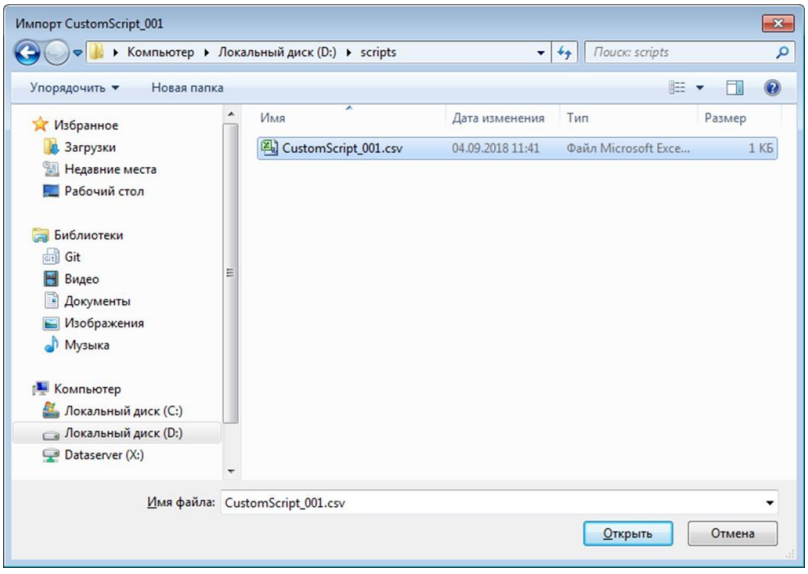

Рисунок 12. Путь к пользовательскому сценарию

4. Выберите файл левой кнопкой мыши и нажмите **Открыть**.
   Данные текущего пользовательского сценария будут заменены на данные импортированного файла.

3.4. Экспорт пользовательского сценария

Для того чтобы экспортировать данные пользовательского сценария из проекта, выполните следующие действия:

1. В дереве проекта правой кнопкой мыши выберите пользовательский сценарий для экспорта.
2. В открывшемся контекстном меню выберите **Экспортировать**:

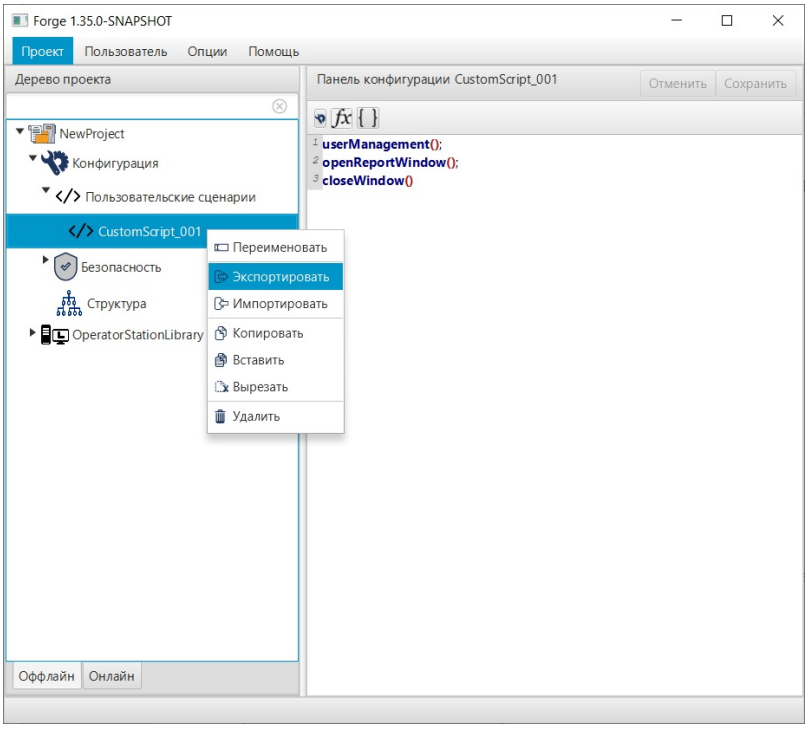

Рисунок 13. Экспорт пользовательского сценария

3. В открывшемся окне **Экспорт <*имя пользовательского сценария*>** укажите имя файла с расширением .csv и путь сохранения.

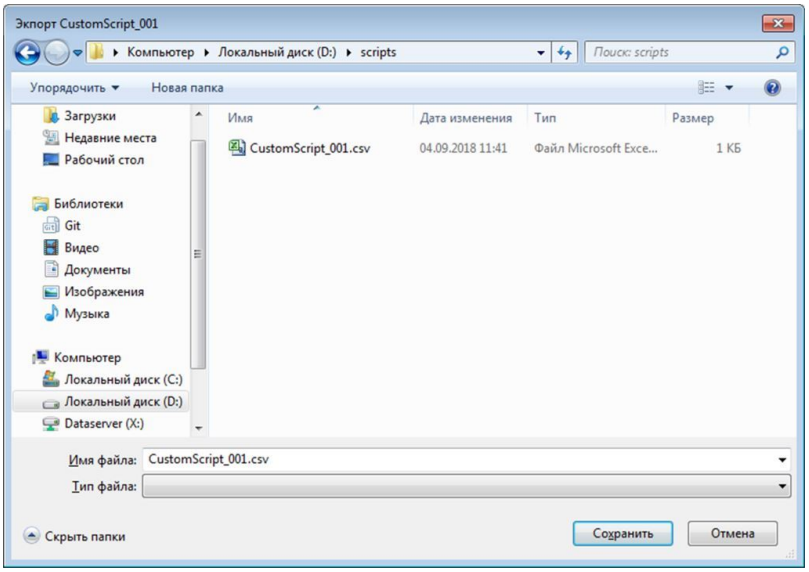

Рисунок 14. Путь сохранения пользовательского сценария

4. Выберите **Сохранить**.
   Файл сохранится в выбранной директории.

3.5. Удаление пользовательского сценария

Для того чтобы удалить пользовательский сценарий из проекта, выполните следующие действия:

1. В дереве проекта откройте вкладки **Конфигурация** > **Пользовательские сценарии**.
2. Правой кнопкой мыши выберите пользовательский сценарий.
3. В открывшемся контекстном меню выберите **Удалить**:

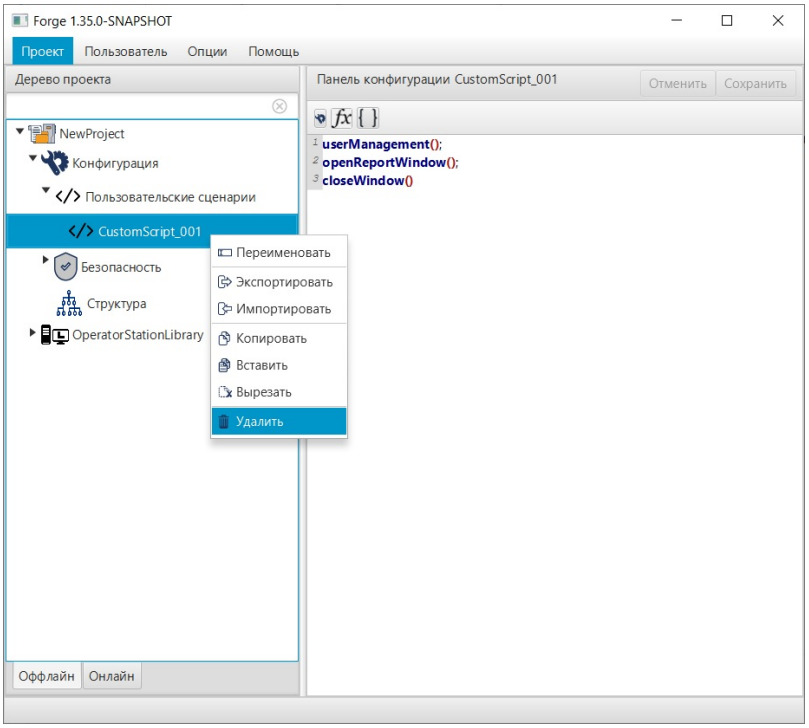

Рисунок 15. Удаление пользовательского сценария

4. В окне подтверждения удаления нажмите **ОК**.
   Пользовательский сценарий будет удален из проекта.

4. Использование псевдонимов в пользовательском сценарии

Параметры можно передавать внутрь функций не напрямую, а с помощью псевдонимов[1].

Пример использования псевдонимов для функции sendEvent:

```
readObjectProperty("${id}", "${property}");
```

При работе с псевдонимами типа String используйте правило установки кавычек:

- если кавычки установлены в сценарии, то они не указываются при подстановке в псевдонимы;
- если кавычки не установлены в сценарии, то ими выделяется значение при подстановке в псевдонимы.

[1] "Руководство по созданию технологического программного обеспечения станции оператора" п. Псевдоним.

5. Библиотека пользовательских сценариев

Библиотека пользовательских сценариев предоставляет описание готовых функций.

5.1. Авторизация и управление пользователями

Вход/выход из системы в режиме исполнения обеспечивает группа функций:

- logIn();
- logOut().

5.1.1. logIn()

Функция входа в систему.

Функция вызывает окно авторизации:

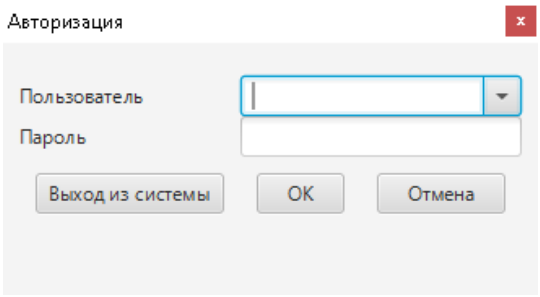

Рисунок 16. Функция logIn()

Поля окна авторизации:

Имя - имя пользователя.

Пароль - пароль для авторизации пользователя.

Если введенные данные верны, пользователь авторизуется. Если данные не верны, отобразится диалоговое окно с сообщением об ошибке.

Для выхода пользователя из системы используется кнопка **Выход из системы**.

5.1.2. logOut()

Функция выхода.

Применяется для сброса авторизации текущего пользователя.

5.2. Отправка сообщений

eventSender обеспечивает отправку сообщений о событии.

Возвращаемое значение

Нет.

Исключительный случай

Нет.

5.2.1. eventSender

Объекту eventSender можно задать получателя, например, ***OPC UA target***, ***Scada Service target***.

Каждый получатель (target) имеет свой набор обязательных аргументов. При отсутствии обязательных аргументов, запись событий не производится.

Аргументы задаются с помощью соответствующих функций. Вызов такой функции возвращает объект eventSender.

С помощью функции `send();` выполняется отправка сообщения указанным получателям (функции addOpcUaTarget, addScadaServiceTarget).

OPC UA target

Обязательные аргументы

message - отправляемое сообщение. Тип String, записывается в кавычках;

priority - приоритет сообщения, изменяется в диапазоне от 1 до 1000. Тип String, записывается в кавычках.

Scada Services target

Обязательные аргументы

message - отправляемое сообщение. Тип String, записывается в кавычках;

severity - критичность сообщения. Может принимать значения: NORMAL, LOW, MEDIUM, HIGH. Записывается в кавычках;

category - шаблон, по которому создается сообщение. Тип String, записывается в кавычках;

Необязательные аргументы

acknowledgable - параметр, описывающий, требуется ли квитировать сообщение:

- acknowledgable (true) - сообщение квитировать;

- acknowledgable (false) - сообщение не квитировать (по умолчанию).

isAlertEvent - параметр, описывающий, требуется ли оповещение о событии:

- isAlertEvent (true) - оповещение есть;
- isAlertEvent (false) - оповещения нет (по умолчанию).

translateEvent - параметр, описывающий, требуется ли переводить сообщение:

- translateEvent (true) - перевод требуется;
- translateEvent (false) - перевод не требуется (по умолчанию).

hideEvent - параметр скрытия события:

- hideEvent (true) - скрыть событие;
- hideEvent (false) - не скрывать событие (по умолчанию).

Пример 1

Пользовательский сценарий, формирующий сообщение для отправки в OPC UA:

```
var sender = eventSender();

sender.message("Message_default").priority("555").addOpcUaTarget().send();
```

Пример 2

Пользовательский сценарий, формирующий сообщение для отправки в ScadaService:

```
var sender = eventSender();

sender.message("Message_default").severity("MEDIUM").category("1").
addScadaServiceTarget().send();
```

Пример 3

Пользовательский сценарий, формирующий сообщение для отправки одновременно и в ScadaService, и в OPC UA:

```
var sender = eventSender();

sender.message("Message_default").priority("555").severity("MEDIUM").
category("1").addScadaServiceTarget().addOpcUaTarget().send();
```

5.3. Чтение и запись данных в теги

Чтение и запись данных в теги обеспечивает группа функций:

- readValue(tagname);

- writeValue(tagname, value).

 **Внимание:** Данные функции не предназначены для работы с битами тегов.

5.3.1. readValue(tagname)

Функция чтения данных из тега.

Аргументы

tagname - имя тега, записывается в кавычках.

Возвращаемое значение

Возвращается значение тега.

Если чтение не удалось - возвращаемое значение null.

Исключительный случай

Нет.

Пример

```
readValue("@{local:Tag_001}");
```

5.3.2. writeValue(tagname, value)

Функция записи данных в тег.

Аргументы

tagname - имя тега, записывается в кавычках;

value - значение для записи в тег.

Возвращаемое значение

В случае успешной записи - true, в случае ошибки - false.

Исключительный случай

При некорректной записи тега или отсутствии связи с сервером ввода/вывода выдается сообщение об ошибке.

Пример

```
writeValue("@{local:Tag_001}", 100);
```

5.4. Чтение и запись данных в свойства визуальных объектов

Чтение и запись данных в в свойства визуальных объектов обеспечивает группа функций:

- readObjectProperty(id, property);
- writeObjectProperty(id, property, value);

Свойства визуальных объектов необходимо указывать приведенным в таблице образом.

Таблица 1. Свойства визуальных объектов для обращения из сценария

Свойство объекта	Имя свойства	Тип данных
Геометрия		
тип фигуры	ARC_TYPE	ENUM
высота	HEIGHT	FLOAT
ширина	WIDTH	FLOAT
начальный угол	START_ANGLE	FLOAT
угловой размер	LENGTH_ANGLE	FLOAT
Графика		
толщина контура	STROKE_WIDTH	FLOAT
цвет заливки	FILL_COLOR	COLOR
цвет контура	STROKE_COLOR	COLOR
тип линии	LINE_TYPE	ENUM
Позиция		
видимость	VISIBILITY	BOOLEAN
x	X	FLOAT
y	Y	FLOAT
Текст		
выравнивание текста	TEXT_ALIGN	STRING
значение текста	TEXT_VALUE	STRING
шрифт	FONT	STRING
Поворот		

Свойство объекта	Имя свойства	Тип данных
угол поворота относительно центра элемента	CENTER_ROTATE_ ANGLE	FLOAT
угол поворота относительно точки элемента	POINTED_ROTATE_ ANGLE	FLOAT
координата **x** точки, вокруг которой вращается элемент	ROTATE_PIVOT_X	FLOAT
координата **y** точки, вокруг которой вращается элемент	ROTATE_PIVOT_Y	FLOAT
Настройки		
Отключение активности элемента	DISABLED	BOOLEAN

5.4.1. readObjectProperty(id, property)

Функция чтения данных из свойств визуальных объектов.

Аргументы

id - идентификатор объекта, записывается в кавычках;

property - наименование свойства объекта, записывается в кавычках.

Возвращаемое значение

Возвращается значение свойства.

Исключительный случай

Если чтение не удалось, возвращаемое значение – null.

Пример 1

```
readObjectProperty("rectangle_001", "width");
```

Пример 2

```
readObjectProperty("${id}", "${property}");
```

5.4.2. writeObjectProperty(id, property, value)

Функция записи данных в свойства визуальных объектов.

Аргументы

id - идентификатор объекта, записывается в кавычках;

property - наименование свойства объекта, записывается в кавычках;

value - значение для записи в свойство объекта.

Возвращаемое значение

При успешной записи - возвращаемое значение true, иначе false.

Исключительный случай

При ошибке – возвращаемое значение false.

Пример 1

```
writeObjectProperty("rectangle_001", "width", 100.0);
```

Пример 2

```
writeObjectProperty("${id}", "${property}", ${value});
```

5.4.3. Особенности функции writeObjectProperty

1. Если у визуального объекта активирована анимация, то запись данных в свойство, связанного с этой анимацией объкта, недоступна. Это правило распространяется на следующие виды анимации.

Таблица 2. Связь анимации со свойством объекта

Анимация визуального объекта	Свойство объекта
анимация заливки	цвет заливки
анимация контура	толщина контура цвет контура
анимация ширины	ширина
анимация высоты	высота
анимация видимости	видимость
анимация текста	значение текста

2. Описание цвета объекта можно задать с помощью следующих цветовых моделей:

- "0xff668840";
- "0xff6688";
- "#ff6688";
- "#f68";
- "rgb(255,102,136)";
- "rgb(100%,50%,50%)";
- "rgba(100%,50%,50%,0.25)";
- "hsl(240,100%,100%)";
- "hsla(120,0%,0%,0.25)".

5.5. Открытие и закрытие окна в режиме исполнения

Открытие и закрытие окна в режиме исполнения обеспечивает группа функций:

- closeWindow();
- closeWindow(name, displayNumber);
- openWindow(name, title, windowParameters, x, y, aliasValues, displayNumber, closeTime, closeWhenCovered);
- openWindow(WindowsSettings).

5.5.1. closeWindow()

Функция закрывает окно, на котором расположен объект анимации.

Примечание: Функция по триггеру не запускается.

Аргументы

Нет.

Возвращаемое значение

Нет.

Исключительный случай

Нет.

5.5.2. closeWindow(name, displayNumber)

Функция закрывает окно заданной мнемосхемы/шаблона.

Обязательные аргументы

name - имя мнемосхемы/шаблона с указанием расширения, в кавычках.

Необязательные аргументы

displayNumber - номер экрана (1-8).

Таблица 3. Значение по умолчанию

Аргумент	Значение по умолчанию
displayNumber	0

Примечание: Указание в качестве аргумента displayNumber экрана, который не используется в системе (в том числе 0), приводит к исполнению пользовательского сценария:

- на текущем экране;
- на первом экране, если сценарий запущен по триггеру.

Возвращаемое значение

Нет.

Исключительный случай

Нет.

Пример 1

Пользовательский сценарий закрывает окно мнемосхемы Mnemonic_001.mnc, принадлежащую второму экрану:

```
closeWindow("Mnemonic_001.mnc", 2);
```

Пример 2

Пользовательский сценарий закрывает окно мнемосхемы Mnemonic_001.mnc на текущем экране:

```
closeWindow("Mnemonic_001.mnc");
```

5.5.3. openWindow(name, title, windowParameters, x, y, aliasValues, displayNumber, closeTime, closeWhenCovered, openIfAlreadyOpened)

Функция открывает окно мнемосхемы/шаблона.

Обязательные аргументы

name - имя мнемосхемы/шаблона с указанием расширения, в кавычках.

Необязательные аргументы

title - заголовок окна, в кавычках;

windowParameters - параметр открытия окна. Содержит константы, которые управляют видом окна;

x - координата x открытия окна;

y - координата y открытия окна;

aliasValues - пара или набор пар псевдоним-значение. Пары отделяются друг от друга знаком точка с запятой: "${Alias1=@{local:TagA}}; ${Alias2=@{local:TagB}}" (подробнее см. п. 4.1. Использование псевдонимов в функции openWindow);

displayNumber - номер экрана (1-8);

closeTime - время (в секундах), по истечении которого окно автоматически закроется;

closeWhenCovered - параметр закрытия статического окна, если оно перекрыто[2]:

- true - закрыть окно, если перекрыто;
- false - не закрывать окно, если перекрыто.

openIfAlreadyOpened - параметр открытия окна по триггеру:

- true - окно будет открываться на текущем экране, даже если уже открыто на другом экране;
- false - если окно уже открыто на другом экране, на него переведется фокус.

Таблица 4. Значения по умолчанию

Аргумент	Значение по умолчанию
title	имя шаблона/мнемосхемы
windowParameters	V_MODAL
x, y	0, 0
aliasValues	"" (пустая строка)
displayNumber	0
closeTime	0 (окно не закрывается)

[2] Перекрытые окна - статические окна (V_STATIC), поверх которых открывается новое статическое окно.

Аргумент	Значение по умолчанию
closeWhenCovered	false (не закрывать, если перекрыто)
openIfAlreadyOpened	false

Примечание 1: Указание в качестве аргумента displayNumber экрана, который не используется в системе (в том числе 0), приводит к исполнению пользовательского сценария на текущем экране.

Причечание 2: Аргумент openIfAlreadyOpened игнорируется, если задан номер экрана.

Константы windowParameters, управляющие видом окна

V_STATIC - статическое окно без обрамления;

V_MODAL - всплывающее окно;

V_MODAL_WITHOUT_CLOSE_BUTTON - всплывающее окно без кнопки закрытия окна;

V_RELATIVE - окно открывается относительно объекта анимации;

V_UNDECORATED - окно открывается без обрамления.

Константы комбинируются оператором "|".

Примечание 2: V_STATIC является взаимоисключающей по отношению к группе констант V_MODAL, V_MODAL_WITHOUT_CLOSE_BUTTON, V_RELATIVE, V_UNDECORATED. V_STATIC не должна применяться в одном пользовательском сценарии с данными аргументами.

Примечание 3: V_RELATIVE и V_UNDECORATED могут применяться только совместно с V_MODAL или V_MODAL_WITHOUT_CLOSE_BUTTON.

Примечание 4: V_RELATIVE не работает в скриптах, исполняемых по триггеру.

При использовании констант некоторые аргументы функции игнорируются:

Таблица 5. Игнорируемые константы

Константа	Игнорируемый аргумент
V_STATIC	title
V_MODAL	-

Константа	Игнорируемый аргумент
V_MODAL_WITHOUT_CLOSE_BUTTON	-
V_RELATIVE	x, y
V_UNDECORATED	title

Возвращаемое значение

Нет.

Исключительный случай

Нет.

Пример 1

Пользовательский сценарий с использованием значений по умолчанию: открыть окно мнемосхемы Mnemonic_001.mnc. Исходные данные:

- заголовок Mnemonic1;
- всплывающее окно;
- координаты окна 0, 0;
- открывается на текущем экране.

```
openWindow("Mnemonic_001.mnc", "Mnemonic1");
```

Пример 2

Пользовательский сценарий: открыть окно мнемосхемы Mnemonic_001.mnc. Исходные данные:

- заголовок игнорируется;
- окно статическое;
- координаты окна 100, 100;
- значение псевдонима ${Alias1} - 200, псевдонима ${Alias2} - @{local:TagA};
- открывается на первом экране.

```
openWindow("Mnemonic_001.mnc", "", V_STATIC, 100, 100, "${Alias1=200};
 ${Alias2=@{local:TagA}}", 1);
```

Пример 3

Пользовательский сценарий: открыть окно мнемосхемы Mnemonic_001.mnc. Исходные данные:

- заголовок игнорируется;
- окно всплывающее, без обрамления;
- координаты окна 100, 100;
- значение псевдонима 200;
- открывается на первом экране.

```
openWindow("Mnemonic_001.mnc", "", V_MODAL|V_UNDECORATED, 100, 100,
 "${Alias=200}", 1);
```

Пример 4

Пользовательский сценарий: открыть окно мнемосхемы Mnemonic_001.mnc. Исходные данные:

- заголовок игнорируется;
- окно всплывающее, без обрамления, открывается относительно объекта анимации;
- координаты игнорируются;
- значение псевдонима 200;
- открывается на втором экране.

```
openWindow("Mnemonic_001.mnc", "", V_MODAL|V_UNDECORATED|V_RELATIVE, 0, 0,
 "${Alias=200}", 2);
```

Пример 5

Пользовательский сценарий: открыть окно мнемосхемы Mnemonic_001.mnc. Исходные данные:

- заголовок игнорируется;
- окно статическое;
- координаты игнорируются;
- значение псевдонима окна ${windowAlias} ссылается на псевдоним скрипта ${scriptAlias};
- открывается на втором экране.

```
openWindow("Mnemonic_001.mnc", "", V_STATIC, 0, 0, "${windowAlias=
${scriptAlias}}", 2);
```

Пример 6

Пользовательский сценарий: открыть окно мнемосхемы Mnemonic_001.mnc, которое будет автоматически закрыто через 3 секунды. Исходные данные:

- заголовок игнорируется;
- окно всплывающее без кнопки закрытия окна;
- координаты окна 100, 100;
- открывается на первом экране;
- закрывается через 3 секунды после открытия.

```
openWindow("Mnemonic_001.mnc","",V_MODAL_WITHOUT_CLOSE_BUTTON,100,100,"",1,3);
```

Пример 7

Пользовательский сценарий: открыть окно мнемосхемы Mnemonic_001.mnc. Исходные данные:

- заголовок игнорируется;
- окно статическое;
- координаты окна 100, 100;
- открывается на текущем экране;
- не закрывается после открытия;
- окно закроется, если будет перекрыто.

```
openWindow("Mnemonic_001.mnc","",V_STATIC,100,100,"",0,0,true);
```

Пример 8

Пользовательский сценарий: открыть окно мнемосхемы Mnemonic_003.mnc на текущем экране по триггеру. Исходные данные:

- заголовок;
- окно статическое;
- координаты окна 0, 0;
- открывается на текущем экране;
- не закрывается после открытия;
- не закрывать окно, если перекрыто;
- окно будет открываться на текущем экране, даже если уже открыто на другом экране.

```
openWindow("Mnemonic_003.mnc","",V_STATIC,0,0,"",0,0,false,true);
```

5.5.4. windowSettings()

С помощью вызова функции windowSettings() можно получить объект WindowSettings, и задать ему свойства для функции openWindow().

Объект WindowSettings содержит свойства, аналогичные аргументам функции openWindow():

- name;
- title;
- x, y;
- aliases;
- displayNumber;
- closeTime;
- closeWhenCovered;
- openIfAlreadyOpened.

Для свойств объекта WindowSettings справедливы те же значения по умолчанию,что и описанные в таблице (см. Таблица 4. Значения по умолчанию).

Помимо вышеперечисленных объект WindowSettings может содержать следующие свойства:

Вид окна:

- modal (true) – модальное окно;
- modal (false) – статическое окно без обрамления (по умолчанию).

Всплывающее окно открывается:

- undecorated (true) - без обрамления;
- undecorated (false) - с обрамлением (по умолчанию).
- relativeToObject (true) - относительно объекта анимации;
- relativeToObject (false) - без привязки в объекту анимации (по умолчанию).
- withoutCloseButton (true) - без кнопки закрытия окна;
- withoutCloseButton (false) - с кнопкой закрытия окна (по умолчанию).

Комбинации этих свойств обрабатываются аналогично комбинациям констант windowParameters (см. п. Константы windowParameters, управляющие видом окна).

Пример 1

Пользовательский сценарий: открыть окно мнемосхемы *test.mnc*. Объект WindowSettings сохраняется в переменной *ws*.

Исходные данные:

- заголовок "Test title";
- открывается на первом экране;
- координаты окна 150, 150;
- закрывается через 10 сек после открытия;
- всплывающее окно с обрамлением и кнопкой закрытия окна, открывается относительно объекта анимации;
- перекрытые окна не закрываются;
- окно не закроется, если будет перекрыто.

```
var ws = windowSettings()

    .title("Test title")
    .name("test.mnc")
    .aliases("")
    .displayNumber(1)
    .x(150)
    .y(150)
    .closeTime(10)
    .undecorated(false)
    .modal(true)
    .relativeToObject(true)
    .withoutCloseButton(false)
    .closeWhenCovered(false);

openWindow(ws); // Вызов сценария openWindow
```

Пример 2

Нет необходимости явно задавать параметры, если их значение совпадает со значением по умолчанию. Таким образом, пример 1 может быть сокращён:

```
var ws = windowSettings()

    .name("test.mnc")
    .title("Test title")
    .displayNumber(1)
    .x(150).y(150)
    .closeTime(10)
    .modal(true)
    .relativeToObject(true);

openWindow(ws);
```

Пример 3

Настроенный объект WindowSettings может быть использован неоднократно:

Предположим, что есть необходимость открыть несколько окон с разным набором псведонимов:

```
var ws = windowSettings()
     .name("TagVisualisation.tpl")
     .displayNumber(1)
     .x(150)
     .y(10)
     .closeWhenCovered(true)
     .aliases("${tag=@{local:Tag001}};${inputDisabled=1}");

openWindow(ws);

    ws.x(400)
    .aliases("${tag=@{local:Tag002}}; ${inputDisabled=1}");

openWindow(ws);

    ws.x(650)
    .aliases("${tag=@{exampleTag}}; ${inputDisabled=0}");

openWindow(ws);
```

5.6. Окно ввода значений

Ввод значения в поле окна в режиме исполнения обеспечивают функции:

- inputWindow(width, height, x, y, title, defaultValue, dataType);
- inputWindow(width, height, title, defaultValue, dataType).

Возвращаемое значение

Функции возвращают значение, введенное в поле, в виде типа данных STRING.

Функции возвращают значение null в следующих случаях:

- отмена ввода (клавиша **“Esc”**);
- переключение фокуса с окна ввода;
- формат вводимых данных не удовлетворяет условию.

Например, если задана проверка данных на целочисленность - INTEGER, то при вводе числа 1,5 функция возвращает null.

5.6.1. inputWindow(width, height, x, y, title, defaultValue, dataType)

Функция обеспечивает открытие окна ввода значения.

Обязательные аргументы

width - ширина окна;

height - высота окна.

Необязательные аргументы

x - координата x открытия окна;

y - координата y открытия окна;

title - заголовок окна, в кавычках;

defaultValue - содержимое окна по умолчанию, в кавычках;

dataType - проверка формата вводимых данных по нажатию клавиши **“Enter“**:

- INTEGER - введенные данные являются целочисленным числом.
- FLOAT - введенные данные являются числом (в т.ч. с десятичным разделителем: точкой или запятой).

Таблица 6. Значения по умолчанию

Аргумент	Значение по умолчанию
title	"" (пустая строка)
x, y	за координаты окна принимаются координаты родительского объекта
defaultValue	"" (пустая строка)
dataType	проверка данных не осуществляется

Пример 1

Пользовательский сценарий с использованием значений по умолчанию: открыть окно ввода Dialog1. Исходные данные:

- заголовок Dialog1;
- за координаты x, y окна принимаются координаты родительского объекта;
- ширина окна равна 250, высота окна равна 100;
- переменная defaultValue имеет значение по умолчанию "" (пустая строка).

```
inputWindow(250, 100, "Dialog1");
```

Пример 2

Пользовательский сценарий: открыть окно ввода Dialog1. Исходные данные:

- заголовок Dialog1;
- координаты окна 100, 100;
- ширина окна равна 250, высота окна равна 100;

- переменная defaultValue имеет значение "0".

```
inputWindow(250, 100, 100, 100,"Dialog1", "0");
```

Пример 3

Пользовательский сценарий: открыть окно ввода без заголовка с проверкой введенных данных. Исходные данные:

- за координаты x, y окна принимаются координаты родительского объекта;
- ширина окна равна 250, высота окна равна 100;
- переменная defaultValue имеет значение "0".
- вводимые данные должны быть целочисленным числом.

```
inputWindow(250, 100, "", "", "", "0", INTEGER);
```

Пример 4

Пользовательский сценарий: открыть окно ввода относительно визуального объекта. Исходные данные сценария inputWindow:

- заголовок окна ввода отсутствует;
- ширина окна ввода равна 100, высота окна равна 25;
- координаты окна ввода вычисляются относительно координат мнемосхемы, объекта и смещения, указанного пользователем;
- переменная defaultValue имеет значение по умолчанию "" (пустая строка).

```
//Вызов функции setValue
setValue("${frameName1}");

//Считывание значений x, y объекта frameName
function setValue(frameName) {
    var x = readObjectProperty(frameName, "x");
    var y = readObjectProperty(frameName, "y");
    //Открытие окна ввода в координатах, учитывающих координаты
    мнемосхемы и объекта
    inputWindow(100, 25, getWindowX() + Number(x) + Number(${offsetX}),
 getWindowY() + Number(y) + Number(${offsetY}));
}
```

где:

- frameName - имя визуального объекта;
- getWindowX(), getWindowY() - координаты мнемосхемы;
- Number(x), Number(y) - координаты визуального объекта;

- Number(${offsetX}), Number(${offsetY}) - смещение по осям, указывается при необходимости.

5.6.2. inputWindow(width, height, title, defaultValue, dataType)

Функция обеспечивает открытие окна ввода значения на месте родительского объекта.

Обязательные аргументы

width - ширина окна;

height - высота окна.

Необязательные аргументы

title - заголовок окна, в кавычках;

defaultValue - содержимое окна по умолчанию, в кавычках.

dataType - проверка формата вводимых данных по нажатию клавиши **“Enter“**:

- INTEGER - введенные данные являются целочисленным числом.
- FLOAT - введенные данные являются числом (в т.ч. с десятичным разделителем: точкой или запятой).

Таблица 7. Значения по умолчанию

Аргумент	Значение по умолчанию
title	"" (пустая строка)
defaultValue	"" (пустая строка)
dataType	проверка данных не осуществляется

Пример 1

Пользовательский сценарий с использованием значений по умолчанию: открыть безымянное окно ввода. Исходные данные:

- заголовок отсутствует;
- ширина окна равна 250, высота окна равна 100;
- переменная defaultValue имеет значение по умолчанию "" (пустая строка)

```
inputWindow(250, 100);
```

Пример 2

Пользовательский сценарий: открыть окно ввода Dialog1. Исходные данные:

- заголовок Dialog1;
- ширина окна равна 250, высота окна равна 100;
- переменная defaultValue имеет значение "1000".

```
inputWindow(250, 100, "Dialog1", "1000");
```

5.7. Координаты окна

Доступ к координатам текущего окна обеспечивают функции:

- getWindowX();
- getWindowY().

getWindowX() позволяет получить координату X текущего окна.

getWindowY() позволяет получить координату Y текущего окна.

Аргументы

Нет.

Возвращаемое значение

Тип данных DOUBLE.

Исключительный случай

Нет.

5.8. Запуск стороннего приложения

Запуск стороннего приложения в системе обеспечивает группа функций:

- runCmd(command,arguments);
- runCmdAndWait(command,arguments).

Возвращаемое значение

Нет.

Исключительный случай

Нет.

5.8.1. runCmd(command,arguments)

Функция запускает стороннее приложение.

Обязательные аргументы

command - команда.

Необязательные аргументы

arguments - аргумент(-ы).

Таблица 8. Значения по умолчанию

Аргумент	Значение по умолчанию
arguments	"" (пустая строка)

Пример 1

Пользовательский сценарий: открыть новое пустое окно приложения "Paint".

```
runCmd("mspaint","");
```

Пример 2

Пользовательский сценарий: открыть определенный файл "test.txt" приложения "Блокнот".

```
runCmd('notepad','C:\\Users\\Desktop\\test.txt');
```

5.8.2. runCmdAndWait(command,arguments)

Функция запускает стороннее приложение и дожидается его завершения.

Обязательные аргументы

command - команда.

Необязательные аргументы

arguments - аргумент(-ы).

Таблица 9. Значения по умолчанию

Аргумент	Значение по умолчанию
arguments	"" (пустая строка)

Пример 1

Пользовательский сценарий: запустить выполнение приложения "Блокнот" и ожидать завершения его работы.

```
runCmdAndWait("notepad","");
```

Пример 2

Пользовательский сценарий: отправить 10 эхо-запросов по протоколу ICMP на ip-адрес 127.0.0.1.

```
runCmd("ping", "127.0.0.1 -n 10");
```

Внимание: Исполнение сценария runCmdAndWait(command,arguments) может привести к зависанию системы.

5.9. Подключение к базе данных

Доступ к базе данных (далее БД) обеспечивает функция подключения к БД по стандарту JDBC[3].

Алгоритм работы для подключения к базе данных:

1. создание подключения (Connection);
2. создание выражения (Statement);
3. SQL-запрос;
4. обработка результата (ResultSet);
5. закрытие соединения.

Connection[4]

Connection представляет собой сеанс подключения к определенной БД.

Вы можете получить объект **Connection** с помощью метода **createSQLConnection()**.

Метод **createSQLConnection (String url)** предназначен для подключения к БД используя заданный url-адрес БД в форме *jdbc: subprotocol: subname*, где:

[3] JDBC — платформенно-независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД.
[4] https://docs.oracle.com/javase/8/docs/api/java/sql/Connection.html

- *jdbc* - имя протокола;
- *subprotocol* - тип БД, к которой вы хотите подключиться. Пример - PostgreSQL, MySQL, ORACLE;
- *subname*- дополнительная информация, такая как имя хоста или имя компьютера, порт и т.д.

По переданному адресу JDBC определяет тип и местоположение БД и возвращает подключение.

Statement[5]

Statement используется для указания SQL-запроса, который должен быть выполнен базой данных.

SQL-запрос

Для выполнения запроса необходимо вызвать метод **executeQuery (String SQL)** класса **Statement**, который возвращает **ResultSet**.

ResultSet[6]

ResultSet представляет собой хранилище для данных, которые вы получаете, выполняя SQL-запрос. Он обеспечивает приложению построчный доступ к результатам запросов в БД.

Доступ к данным в объекте **ResultSet** осуществляется с помощью курсора. Во время обработки запроса **ResultSet** поддерживает указатель на текущей обрабатываемой строке. Приложение последовательно перемещается по результатам, пока они не будут все обработаны или не будет закрыт **ResultSet**.

Закрытие соединения

По окончанию обработки следует закрыть **Statement** и **ResultSet**. После чего необходимо закрыть **Connection** - подключение к БД, и таким образом завершить работу с программой.

Пример

Пользовательский сценарий, формирующий таблицу данных, извлеченных из удаленной БД.

---

[5] https://docs.oracle.com/javase/8/docs/api/java/sql/Statement.html
[6] https://docs.oracle.com/javase/8/docs/api/java/sql/ResultSet.html

Исходные данные: url-адрес БД - jdbc:postgresql://localhost/testDB?user=testUsers&password=testPwd, где testDB - наименование БД, testUsers - имя пользователя, testPwd - пароль.

```
//Создание соединения:
var connection = createSQLConnection("jdbc:postgresql://localhost:5432/testDB?
user=testUsers&password=testPwd");

//Проверка соединения:
if (connection) {
    writeValue("@{LocalString_001}", 'Соединение с БД установлено');
} else {
    writeValue("@{LocalString_001}", 'Соединение с БД не установлено');
}

//Обработка ошибок (try....catch):
try {
    //Создание Statement с возможностями
    перемещения курсора и записи в БД:
    var statement = connection.createStatement(1005, 1008);

    //Создание Result Set:
    var rs = statement.executeQuery("SELECT * FROM testTable");

    //Выборка из Result Set (первая строка):
    rs.first();
    writeValue("@{LocalInt_001}", rs.getInt("Number"));
    writeValue("@{LocalString_002}", String(rs.getString("Message")));

    //Положение курсора:
    writeValue("@{LocalInt_002}", rs.getRow());

//Вывод ошибок:
} catch(err) {
    writeValue("@{LocalString_003}", err.message);
}

//Закрытие соединения:
rs.close();
statement.close();
connection.close();
```

5.9.1. Настройка resultSetModes

Константы **resultSetModes** для настройки режима работы объекта аналогичны константам **java.sql.ResultSet**.

Таблица 10. Константы resultSetModes

Константа	Описание
CLOSE_CURSORS_ AT_COMMIT	Объекты ResultSet, созданные в текущей транзакции (которые уже открыты), будут закрыты после того, как транзакция будет сохранена / зафиксирована

Константа	Описание
CONCUR_READ_ONLY	Создаёт экземпляр ResultSet только для чтения. Устанавливается по умолчанию
CONCUR_UPDATABLE	Создаёт экземпляр ResultSet, который может изменять данные
FETCH_FORWARD	Строки в наборе ResultSet будут обрабатываться в прямом/последовательном направлении; от первой до последней
FETCH_REVERSE	Строки в наборе ResultSet будут обрабатываться в обратном направлении: от последней к первой
FETCH_UNKNOWN	Порядок обработки строк в наборе ResultSet неизвестен/ неопределен
HOLD_CURSORS_ OVER_COMMIT	Объекты ResultSet, созданные в текущей транзакции (которые уже открыты), будут оставаться открытыми после того, как транзакция будет сохранена/ зафиксирована
TYPE_FORWARD_ONLY	Указатель двигается только вперёд по множеству полученных результатов
TYPE_SCROLL_ INSENSITIVE	Указатель может двигаться вперёд и назад и не чувствителен к изменениям в БД, которые сделаны другими пользователями после того, как ResultSet был создан
TYPE_SCROLL_ SENSITIVE	Указатель может двигаться вперёд и назад и чувствителен к изменениям в БД, которые сделаны другими пользователями после того, как ResultSet был создан

Пример 1

Результатом обработки является набор данных, в котором указатель может двигаться вперёд и назад, и не чувствителен к изменениям в БД (не будет отображать изменения, сделанные другими), и доступен для редактирования

(можно изменить значение внутри `resultSetModes`, и оно автоматически отразится в соответствующем столбце нужной строки БД).

```
var connection = createSQLConnection("jdbc:postgresql://localhost/testDB?
user=testUsers&password=testPwd");

var statement = connection.createStatement(
    resultSetModes().get("TYPE_SCROLL_INSENSITIVE"),
    resultSetModes().get("CONCUR_UPDATABLE"));

var rs = statement.executeQuery("SELECT a, b FROM TABLE2");
```

5.10. Вывод логов

Вывод сообщений о событиях с указанием уровня события в соответствующий файл логов обеспечивается функцией logger(logLevel, logMessage).

Обязательные аргументы

logMessage - сообщение.

Необязательные аргументы

logLevel - уровень события.

Таблица 11. Значения по умолчанию

Аргумент	Значение по умолчанию
logLevel	"WARN"

Таблица 12. Уровни событий

Уровень события	Описание
FATAL	ошибка, которая может привести к прерыванию работы приложения
ERROR	ошибка в приложении
WARN	сообщение об ошибке или нестандартной ситуации
INFO	информационное сообщение
DEBUG	сообщение о событии отладки приложения
TRACE	детализированное сообщение о событии отладки приложения

Возвращаемое значение

Нет.

Исключительный случай

Нет.

Пример 1

```
logger("INFO", "Message_INFO");
```

Результат выполнения сценария:

```
14:42:14,854 -> INFO [JavaFX Application Thread] logger.ScriptLogger:accept:19
 - Message_INFO
```

Пример 2

```
logger("Message_default");
```

Результат выполнения сценария:

```
14:42:14,852 -> WARN [JavaFX Application Thread] logger.ScriptLogger:accept:23
 - Message_default
```

5.11. Чтение данных Modbus

5.11.1. readModbusArray(channel, tagName, left, right)

Функция readModbusArray(channel, tagName, left, right) позволяет считать заданный диапазон тегов из одного массива заданного канала Modbus.

Аргументы

channel - имя канала Modbus, из которого будут прочитаны значения тегов.

tagName, left, right - будут прочитаны значения тегов от tagName[left] до tagName[right] включительно, где left, right - индексы первого и последнего тега.

Возвращаемое значение

Объект readingResults.

Пример 1

Исходные данные: Требуется считать значения тегов Tag_001[1], Tag_001[2], Tag_001[3] из канала ModbusTCPChannel_001.При условии что:

- ${L} = 1, ${R} = 3;
- ${tagName} = Tag_001.

```
var results = readModbusArray("ModbusTCPChannel_001", "@{local:${tagName}}",
 ${L}, ${R});

if (!results.isGood()) {
writeValue("@{read_array_msg}", results.getMessage());

}
else {
    for (var i = 0; i <= ${R} - ${L}; ++i) {
        var index = ${L} + i;
        var remoteTagName = "@{local:${tagName}[" + index + "]}";
        var value = results(remoteTagName);
        var localTagName = "@{array_" + i + "}";

        writeValue(localTagName, value);
    }
}
```

Примечание:

Результаты выполнения функции возвращаются в объекте readingResults. Вызов нижеуказанных функций позволяет получить детали выполнения пользовательского сценария readModbusArray(channel, tagName, left, right):

`results.isGood();` - получение статуса выполнения команды;

`results.getMessage();` - получение сообщения об ошибке, если команда выполнена с ошибкой;

`results(tagNameWithIndex);` - получение прочитанных значений тегов, если чтение прошло без ошибок.

5.11.1.1. Квитирование сигнализации

Квитирование определенной сигнализации обеспечивает функция acknowledgeAlarm(alarmName).

Возвращаемое значение

При успешном квитировании - возвращаемое значение true, иначе false.

Исключительный случай

Если чтение не удалось, возвращаемое значение – null.

Аргументы

alarmName - имя сигнализации, в кавычках.

Пример

Пользовательский сценарий квитирует сигнализацию:

```
acknowledgeAlarm("Alarm_001");
```

Скрипты acknowledgeSelected(alarmPadId) и acknowledgeVisible(alarmPadId) - оба принимают objectId планшета сигнализаций мнемосхемы, на которой они запускаются. Функционал эквивалентен кнопкам "Квитировать" и "Квитировать всё" этого самого планшета сигнализаций.

5.12. Квитирование сигнализации

Квитирование определенной сигнализации обеспечивает функция acknowledgeAlarm(alarmName).

Возвращаемое значение

При успешном квитировании - возвращаемое значение true, иначе false.

Исключительный случай

Если чтение не удалось, возвращаемое значение – null.

Аргументы

alarmName - имя сигнализации, в кавычках.

Пример

Пользовательский сценарий квитирует сигнализацию:

```
acknowledgeAlarm("Alarm_001");
```

Скрипты acknowledgeSelected(alarmPadId) и acknowledgeVisible(alarmPadId) - оба принимают objectId планшета сигнализаций мнемосхемы, на которой они запускаются. Функционал эквивалентен кнопкам "Квитировать" и "Квитировать всё" этого самого планшета сигнализаций.

5.13. Подавление (откладывание) сигнализации

Подавление (откладывание) до первого изменения определенной сигнализации обеспечивает функция oneShotShelveAlarm(alarmName):

Аргументы

alarmName – имя сигнализации, записывается в кавычках.

Возвращаемое значение

При успешном выполнении – возвращаемое значение true, иначе false.

Исключительный случай

Если отложить не удалось, возвращаемое значение – null.

Пример

Пользовательский сценарий откладывает сигнализацию:

```
oneShotShelveAlarm("Alarm_001");
```

Функция oneShotShelveGroup(groupName) служит для подавления группы сигнализаций. Функция принимает один параметр – имя группы. В качестве имени группы выступает имя структурной единицы сигнализации.

Скрипты oneShotShelveSelected(alarmPadId) и oneShotShelveVisible(alarmPadId) – оба принимают objectId планшета сигнализаций мнемосхемы, на которой они запускаются. Функция oneShotShelveSelected(alarmPadId) откладывает выделенную сигнализацию, а oneShotShelveVisible(alarmPadId) все видимые.

Функции shelveAlarm(alarmName), shelveSelected(alarmPadId) и shelveVisible(alarmPadId) предназначены для подавления (откладывания) сигнализаций:

Функция shelveAlarm(alarmName) обеспечивает подавление (откладывание) определенной сигнализации.

Скрипты shelveSelected(alarmPadId) и shelveVisible (alarmPadId) – оба принимают objectId планшета сигнализаций мнемосхемы, на которой они запускаются. Функция shelveSelected(alarmPadId) откладывает выделенную сигнализацию, а shelveVisible(alarmPadId) все видимые.

Функции timedShelveAlarm(alarmName, durationSeconds), timedShelveGroup(groupName, durationSeconds), timedShelveSelected(alarmPadId, durationSeconds) и timedShelveVisible(alarmPadId, durationSeconds) предназначены для откладывания сигнализаций на время. У функций появляется дополнительный параметр – количество секунд, на которые нужно отложить сигнализацию.

Функции unshelveAlarm(alarmName), unshelveGroup(groupName), unshelveSelected(alarmPadId) и unshelveVisible(alarmPadId) предназначены для снятия подавления (откладывания) у ранее подавленных сигнализаций.

Прим.: В качестве имени группы выступает имя структурной единицы. Имя структурной единицы (группы) присваивается всему блоку или отдельной сигнализации в блоке.

5.14. Управление звуками сигнализации

Управление звуками обеспечивают следующие функции:

- acknowledgeSound();
- beep();
- muteSound(mute);
- playSound(name).

5.14.1. acknowledgeSound()

Функция прекращает воспроизведение текущего звука. Звук возобновится при следующей активной сигнализации.

Аргументы

Нет.

Возвращаемое значение

Нет.

Исключительный случай

Нет.

Пример

Пользовательский сценарий, прекращающий воспроизведение текущего звука:

```
acknowledgeSound();
```

5.14.2. beep()

Функция воспроизводит системный звук "beep". Звук воспроизводится только через встроенный динамик устройства и не использует внешние колонки.

Аргументы

Нет.

Возвращаемое значение

Нет.

Исключительный случай

Нет.

Пример

Пользовательский сценарий, проигрывающий короткий системный звук "beep":

```
beep();
```

5.14.3. muteSound(mute)

Функция управляет включением или отключением звука в динамиках. Эта функция позволяет временно отключить или включить звук в зависимости от переданного параметра.

Обязательные аргументы

mute - отключение звука, тип Boolean;

muteSound(true) – отключение звука;

muteSound(false) – включение звука.

Необязательные аргументы

Нет.

Возвращаемое значение

Нет.

Исключительный случай

Нет.

Пример

Пользовательский сценарий, управляющий включением/отключением звука:

```
// Отключить звук
muteSound(true);

// Включить звук
muteSound(false);
```

5.14.4. playSound(name)

Функция воспроизводит звуковой файл из системной библиотеки звуков. Звук проигрывается через динамики, указав путь к файлу звука.

Обязательные аргументы

name – полный путь к звуковому файлу в системной библиотеке звуков. Тип String, записывается в кавычках.

Необязательные аргументы

Нет.

Возвращаемое значение

Нет.

Исключительный случай

Нет.

Пример

Пользовательский сценарий воспроизводит звуковой файл из библиотеки звуков:

```
playSound("OS01.sounds_library.SoundsFolder_001.Alert");
```

5.15. Окно просмотра отчетов

Функция openReportWindow() вызывает в среде исполнения окно просмотра отчетов, которое позволяет сформировать отчет.

Возвращаемое значение

Нет.

Исключительный случай

Нет.

Пример

Пользовательский сценарий вызывает окно просмотра отчетов:

```
openReportWindow();
```

5.16. Закрытие окна режима исполнения

Функция exitVision() закрывает окно режима исполнения.

Возвращаемое значение

Нет.

Исключительный случай

Нет.

Пример

Пользовательский сценарий закрывает окно режима исполнения:

```
exitVision();
```

5.17. Вызов методов OPC сервера

Функции DLL из скриптов NaftaProcess вызываются с помощью скрипта callOpcMethod(String objName, String methodName, Object... args).

Обязательные аргументы

objName - имя библиотеки DLL;

methodName - имя вызываемоей функции в системе.

Опциональные аргументы

Набор значений аргументов функции в произвольном количестве, перечисляются через запятую:

```
callOpcMethod(lib_name, func_name); // 0 аргументов
callOpcMethod(lib_name, func_name, arg1, arg2, arg3); // 3 аргумента
```

Возвращаемое значение

Объект MethodCallResult, у этого объекта можно вызвать следующие функции:

- isGood() — true/false, функция вызвана успешно или с ошибкой

- getMessage() — описание ошибки при вызове функции, если isGood вернул false

- getValue(int index), getValue() — если функция вызвана успешно, то получить результат вызова; несколько значений из функции может быть получено, если у неё не VOID возвращаемое значение и/или есть аргументы типа STRING, отмеченные как buffer(sz), вызов без параметров "getValue()" равносилен вызову "getValue(0)"

Пример 1

```
// Допустим, функция func_name имеет три аргумента типов DOUBLE, INT16 и STRING,
// возвращает BOOL как результат функции и пишет какое-то значение в третий
// аргумент, имеющий свойство buffer(10)

let res = callOpcMethod("DLL1", "func_name", 100.321, 3, "");
```

```
if (res.isGood()) {
    // по описанию функции func_name у неё два возвращаемых значения:
    // 0) BOOL #ret — успешность операции
    // 1) STRING #2 — строка

    if (res.getValue(0)) { // если операция выполнена успешно
       let str = res.getValue(1); // присвоить полученное из функции значение
                                  // строки переменной str
       writeValue("@{LocalTag_001}", str); // записать полученное значение
                                           // в локальный тег
    }
}
// номер, по которому можно получить это значение из MethodCallResult, и номер
// в списке аргументов — разные номера
```

5.18. Доступ к фильтрам сигнализации

С помощью вызова функции alarmFilterSettings() можно получить объект AlarmFilterSettings, и задать ему значения фильтров для планшета сигнализаций и/или журнала событий.

5.18.1. alarmFilterSettings()

alarmFilterSettings() имеет следующие методы для задания значения фильтров:

setActive - параметр, который задает состояние активности сигнализации:

- setActive(true) - сигнализация активна;
- setActive(false) - сигнализация не активна (по умолчанию).

setAcked - параметр, который задает состояние квитированности сигнализации:

- setAcked(true) - сигнализация квитирована;
- setAcked(false) - сигнализация не квитирована (по умолчанию).

setShelved - параметр, который задает фильтр подавления сигнализации:

- setShelved(true) - активна фильтрация подавления сигнализации, отображаются все подавленные сообщения;
- setShelved(false) - активна фильтрация подавления сигнализации, отображаются все неподавленные сообщения.

setUserId - пользователь, квитировавший сигнализацию. Тип STRING, записывается в кавычках;

setMessage - описание сигнализации. Тип STRING, записывается в кавычках;

setSource - источник сигнализации. Тип STRING, записывается в кавычках;

setName - имя сигнализации. Тип STRING, записывается в кавычках;

setPriority - приоритет. Тип STRING, записывается в кавычках;

setStructureUnit - структурная единица. Тип STRING, записывается в кавычках.

Прим.: Объект AlarmFilterSettings изначально создаётся с пустыми настройками - все фильтры выключены и пустые. Если фильтру задаётся значение с помощью соответствующего метода, то он автоматически становится включённым.

Пример 1

Задаём все фильтры:

```
alarmFilterSettings()
    .setActive(false)
    .setAcked(false)
    .setShelved(false)
    .setUserId("${usedId}")
    .setMessage("${message}")
    .setSource("${src}")
    .setName("${name}")
    .setPriority("${prior}")
    .setStructureUnit("${structure}")
    .applyTo("${tableId}");
```

Пример 2

Сбрасываем все фильтры:

```
alarmFilterSettings()
    .applyTo("${tableId}");
```

Пример 3

Задаём фильтры по состояниям активности и квитированности:

```
let afs = alarmFilterSettings();

afs.setActive(true);
afs.setAcked(true);
afs.applyTo("alarmTable_001");
```

5.18.2. alarmFilterSettings(filterSource)

alarmFilterSettings(filterSource) - в этом варианте скрипта изначально создаётся не пустой фильтр, а копируются настройки фильтра указанной таблицы.

Пример 1

Добавляем к текущим фильтрам таблицы ${tableId} фильтр по структурной единице:

```
alarmFilterSettings("${tableId}")
    .setStructureUnit("${structureUnit}")
    .applyTo("${tableId}");
```