Руководство по созданию скриптов (программирование)

Интеллект 5.0 (русский)

Обновлено 06.02.2025

Содержание

1 Руководство по созданию скриптов. Введение

2 Объект Программа. Программирование с использованием встроенного языка ПК Интеллект

3 Объект Скрипт. Программирование с использованием языка JScript

4 Описание событий и реакций объектов системы

5 Описание объектной модели в программном комплексе Интеллект

6 Заключение

7 Руководство по созданию скриптов. Введение

Программирование с помощью скриптов используется, если для реализации какого-либо сценария работы ПК *Интеллект* недостаточно его интерфейсных настроек объектов или возможностей Макрокоманд (подробнее о вариантах настроек взаимосвязей объектов в таблице ниже).
В ПК *Интеллект* возможны два варианта написания скриптов:

1. **На встроенном языке программирования**. Для реализации используется системный объект **Программа** вкладки Программирование – см. Объект Программа. Программирование с использованием встроенного языка ПК Интеллект.
2. **На языке JScript**. Для реализации используется системный объект **Скрипт** вкладки Программирование – см. Объект Скрипт. Программирование с использованием языка JScript.

Оба способа работающие, но объект **Программа** является устаревшим и больше не развивается. Мы рекомендуем использовать объект **Скрипт** и язык JScript для написания скриптов.

В этом руководстве:

- описаны настройки для обоих объектов,
- синтаксис скриптов и их отладка,
- для каждого языка приведены примеры скриптов.

В скриптах используются:

- события, реакции и команды объектов ПК *Интеллект*. Основные из них описаны в разделе Описание событий и реакций объектов системы;
- объекты Core, MsgObject, Event и их встроенные методы, описанные в разделе Описание объектной модели в программном комплексе Интеллект.

7.1 Способы настройки взаимосвязей между объектами в программном комплексе Интеллект

Функциональные возможности программного комплекса *Интеллект* основаны на логическом взаимодействии между объектами. Общие сведения о способах настройки логических взаимосвязей:

Способ настройки логической взаимосвязи	**Описание**	**Реализация**	**Пример**
Панели настройки объектов системы	Базовая настройка взаимодействия между объектами системы	Реализуется с использованием возможностей объектов системы – см. Конфигурирование и настройка программного комплекса Интеллект	Настройка отображения видео с **Камеры** в интерфейсном окне **Монитор**
Макрокоманда	Настройка простых взаимосвязей между объектами, если базовых настроек объектов недостаточно	Реализуется с использованием объекта **Макрокоманда** – см. Создание и использование макрокоманд	Настройка включения исполнительного устройства (реле) при замыкании луча
Программа	Настройка комплексных взаимосвязей между объектами, если возможностей объекта **Макрокоманда** недостаточно	Реализуется на базе объекта **Программа** в виде кода на встроенном в ПК *Интеллект* языке программирования – см. Объект Программа. Программирование с использованием встроенного языка ПК Интеллект	Требуется каждые 15 минут возвращать поворотные камеры в исходное положение и делать снимок
Скрипт		Реализуется на базе объекта **Скрипт** в виде кода на языке JScript – см. документ Объект Скрипт. Программирование с использованием языка JScript	

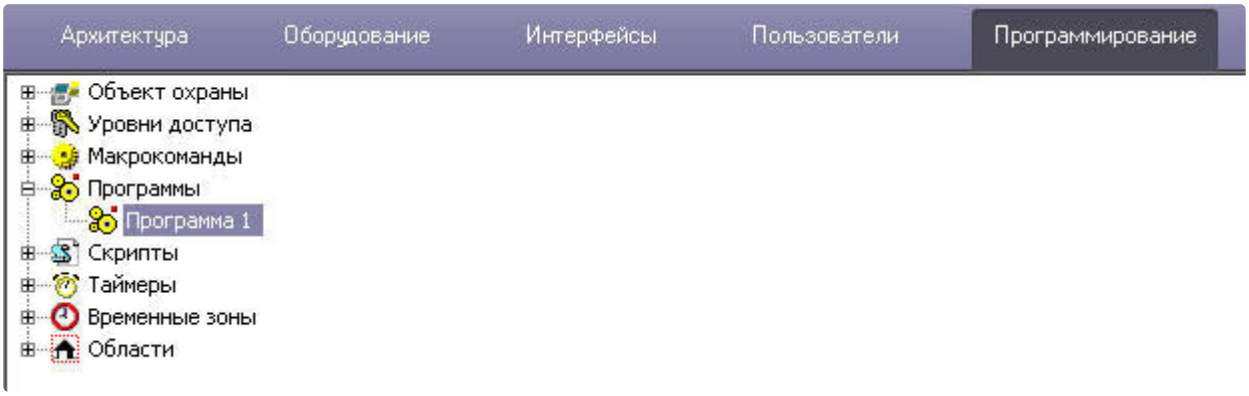
8 Объект Программа. Программирование с использованием встроенного языка ПК Интеллект

8.1 Инструментарий программирования

8.1.1 Системный объект Программа

Системный объект **Программа** предназначен для инициализации в ПК *Интеллект* программы, разработанной на собственном языке программирования ПК *Интеллект*, и задания параметров ее выполнения.

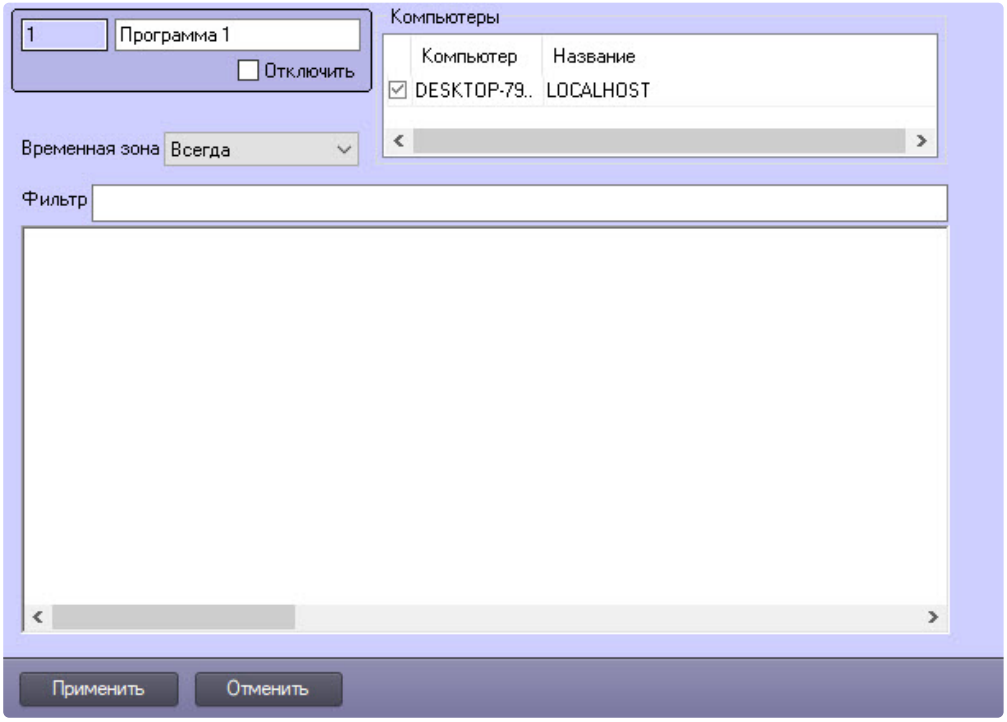
Системный объект **Программа** создается на базе объекта **Программы** на вкладке **Программирование** диалогового окна **Настройка системы**.



Внимание!

Создание большого количества (более 100) объектов **Программа** на одном компьютере может привести к нестабильной работе системы.

Панель настройки системного объекта **Программа** представлена на рисунке:



В панели настройки системного объекта **Программа** указываются временная зона выполнения программы и компьютеры (ядра), на которых требуется выполнять программу.

> **Примечание.**
>
> Для того, чтобы установить флажки напротив всех компьютеров, необходимо выделить ячейку в столбце с флажками и нажать Ctrl+A. Для снятия всех флажков необходимо выделить ячейку и нажать Shift+A.

Для предварительной фильтрации обрабатываемых программой событий следует задать значение в поле **Фильтр**. Формат фильтра – ТИП|ID|СОБЫТИЕ, разделенные точкой с запятой. Например, фильтр CAM||MD_STOP;CAM||MD_START позволит отфильтровать события **Тревога** и **Конец тревоги** от всех объектов **Камера**.

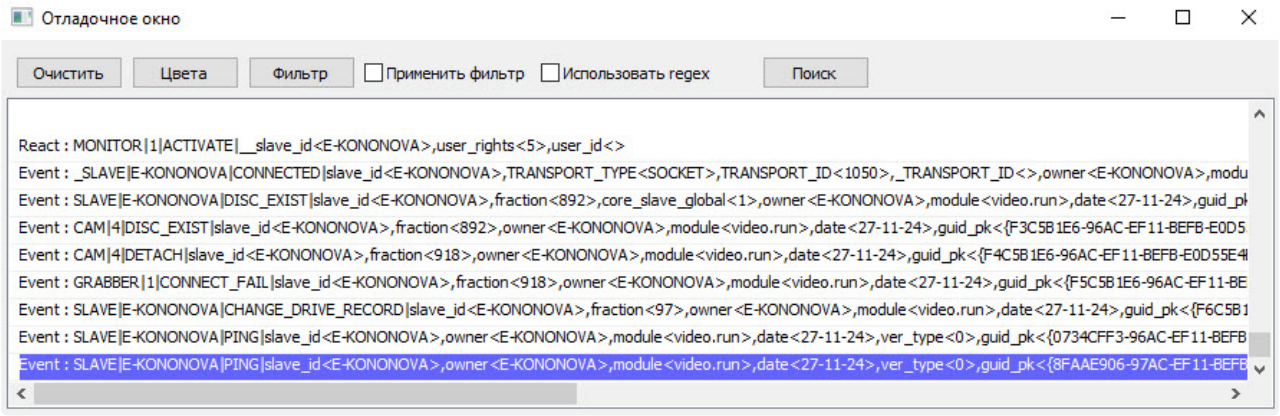
На панели настройки системного объекта **Программа** размещен текстовый редактор для написания и редактирования кода программы.

В текстовом редакторе на панели настроек системного объекта **Программа** есть возможность отмены действия и повтора с помощью горячих клавиш. Для отмены действия нажмите **Alt+Backspace**, для повтора – **Ctrl+Y**.

8.1.2 Отладочное окно

Отладочное окно программного комплекса *Интеллект* предназначено для просмотра сведений обо всех событиях, регистрируемых в системе.

Вызов **Отладочного окна** выполняется с помощью команды **Отладочное окно** из меню **Выполнить** главной панели управления.
Отладочное окно программного комплекса *Интеллект* отображается в нижней части экрана.



По умолчанию **Отладочное окно** открывается в режиме отладки Debug 4. Изменить режим отладки **Отладочного окна** можно с помощью утилиты *tweaki.exe* (см. раздел Отладочное окно документа Объект Скрипт. Программирование с использованием языка JScript) или с помощью ключа реестра **DebugLevel** (подробнее см. Справочник ключей реестра, подробнее о работе с реестром см. Работа с системным реестром ОС Windows).

8.1.3 Синтаксический анализатор

Встроенный синтаксический анализатор позволяет отслеживать правильность написания основных зарегистрированных слов, таких как OnEvent, DoReact, OnTime, Wait, Sleep и др. Эти зарегистрированные слова отмечаются черным цветом в поле текста программы. Следует отметить, что за правильностью написания параметров команд анализатор не следит, и нужно быть особенно внимательным в этих случаях.

```
OnEvent("MACRO","2","RUN")
{[
fn="D:\Intellect\Bmp\Person\1.bmp";

DoReact("MONITOR","1","EXPORT_FRAME","cam<1>,file<"+fn+">");
DoReact("DIALOG","operator","CLOSE_ALL");
Sleep(500);
DoReact("DIALOG","operator","RUN");
]}

OnEvent("MACRO","3","RUN")
{[
fn="D:\Intellect\Bmp\Person\1.bmp";

DoReact("MONITOR","1","EXPORT_FRAME","cam<2>,file<"+fn+">");
```

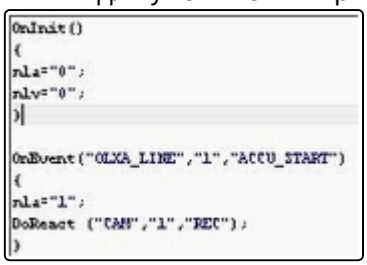
Для изменения размера шрифта используйте сочетания клавиш:

- **CTRL** и **+** для увеличения шрифта

```
OnInit()
{
n1a="0";
n1v="0";
}

OnEvent("OLXA_LINE","1","ACCU_START")
{
n1a="1";
DoReact ("CAM","1","REC");
}
```

- **CTRL** и **-** для уменьшения шрифта



8.1.4 Рекомендуемый порядок написания программ

На странице:

1. Постановка общей задачи.
2. Разбитие задачи на подзадачи.
3. Написание подзадач и их отладка.
4. Поиск и исправление ошибок.

8.1.4.1 Постановка общей задачи

Нужно четко представлять, что должно происходить в системе при определенных событиях. Определить ID устройств, участвующих в генерации событий и действий.

8.1.4.2 Разбитие задачи на подзадачи

Если задача подразумевает обработку нескольких различных событий, то имеет смысл четко представить действия системы на каждое из этих событий. По возможности нужно исключить возможность бесконечного зацикливания выполнения скриптов, т.е исключить всяческие рекурсивные действия, если конечно они не предусматривают выполнение поставленной задачи.

8.1.4.3 Написание подзадач и их отладка

Наиболее сложным в написании скриптов является написание списка действий с возможным использованием логических и циклических операций. По опыту эта часть программирования наиболее долго отлаживается. Зачастую генерация события, требующая обработки, является не очень удобной, тем более на реальном объекте – например, срабатывание пожарного датчика или движение по камере, достаточно удаленной от места программирования (от сервера с ядром системы). В этом случае рекомендуется на этапе отладки действий генерировать событие вручную, самое удобное – это запуск пустой макрокоманды. После отладки тела скрипта в событие вместо запуска пустой макрокоманды подставляется реальное событие. Кроме того можно проверить и, наоборот, убедиться в правильности написания реального события, не запуская списка действий, вставив вместо списка действий запуск пустой макрокоманды и посмотреть ее выполнение в отладочном окне.

8.1.4.4 Поиск и исправление ошибок

Встроенный синтаксический анализатор на этапе запуска программы проверяет правильность написания названий функций, но не проверяет правильность синтаксиса программы (расстановки ключевых символов: запятых, точек с запятой, вложенность скобок). Чтобы отследить ошибки в программе, если они есть, необходимо активировать режим отладки **Debug 4** (см. Включение и настройка режима отладки программного комплекса Интеллект). В случае наличия синтаксических ошибок на этапе исполнения тела программы отобразится окно **Критические ошибки**, в котором будут перечислены названия функций с неверным синтаксисом и другая отладочная информация.

Примечание

В том случае, если синтаксис программы правильный, но программа не работает или работает с ошибками, рекомендуется переписать программу в виде скрипта на языке JScript (см. Объект Скрипт. Программирование с использованием языка JScript).

8.2 Описание синтаксиса

Скрипт состоит из набора процедур.

Все операторы, выполняемые внутри процедур, формируются в блоки {..}.

Если нужно вставить комментарий, то перед комментарием требуется поставить спецсимволы //.

8.2.1 Описание переменных

Все переменные, используемые в системе – строковые.

Для сравнения строковых переменных и значений используется функция: bool strequal (строка1,строка2). Функция "strequal" возвращает значение, отличное от нуля, если строки равны (см. раздел Описание функций).

Для произведения целочисленных действий используется функция: str(строка1) (см. раздел Описание функций).

8.2.2 Описание процедур

8.2.2.1 Стандартные процедуры

Существуют 3 стандартные процедуры, которые могут быть выполнены при возникновении соответствующего события:

1. **OnInit**() – используется для инициализации переменных (задания первоначальных значений), которые будут в дальнейшем использоваться при выполнении скриптов. Выполняется до старта всех модулей системы. Рекомендуется использовать один вызов процедуры на все существующие скрипты.
   Пример использования:

```
OnInit(){
                flag=1;
                num=8;   //на старте системы будут проинициализированы переменные
}
```

2. **OnTime**(день недели (1-7), день-месяц-год, часы, минуты, секунды) – запуск в определенный момент времени.

```
OnTime(W,D,X,Y,H,C,S)
{
//W - день недели (0 - понедельник, 6 - воскресенье);
//D - дата в формате "число-месяц-год", 16 августа 2001 года это "16-08-01"
//X,Y - зарезервировано
                //H - час
                //C - минуты
                //S - секунды
                // ВЫПОЛНЯЯ СРАВНЕНИЕ С ПАРАМЕТРАМИ, ДАЛЕЕ УКАЗЫВАЕТСЯ
ДЕЙСТВИЕ
}
```

   Примеры использования:

```
OnTime(W,"16-08-01",X,Y,"11","11","30")
{
                // помещенный здесь код сработает 16 августа 2001 года в 11 часов 11
минут 30 секунд
}
```

```
OnTime(W,D,X,Y,"11","11","30")
```

```
{
                // помещенный здесь код сработает каждый день в 11 часов 11 минут 30
секунд
}
```

```
OnTime(W,"16-08-01",X,Y,H,C,S)
{
                // помещенный здесь код будет срабатывать 16 августа 2001 года
                // каждую секунду
}
```

```
OnTime(W,"16-08-01",X,Y,"11","11",S)
{
                // помещенный здесь код будет срабатывать 16 августа 2001 года
                // с 11 часов 11 минут по 11 часов 12 минут каждую секунду
}
```

```
OnTime("0",D,X,Y,"21","0","0")
{
                // помещенный здесь код будет срабатывать каждый понедельник
                // в 21 часов 00 минут 00 секунд
}
```

3. **OnEvent**(тип источника, номер, событие) – запуск по определенному событию от объекта системы. Основная процедура при написании скриптов.
   Примеры использования:

```
OnEvent("GRAY","1","ON")
{
                // Выполнится при замыкании луча №1
}
```

```
OnEvent("CAM","12","MD_START")
{
                // Выполнится при срабатывании детектора движения камеры №12
}
```

Каждая процедура, имеющая параметры, может встречаться в коде много раз с различными параметрами. При возникновении события система выполнит те из них, параметры которого совпадут с параметрами возникшего события.

Параметр процедуры может быть определенным или нет. В первом случае его значение берется в кавычки, в последнем случае параметр обозначается латинскими буквами, и процедура будет выполнена для всех событий, для которых его можно определить.

Примеры использования:

```
OnEvent("GRAY","1","ON")        // Выполнится при замыкании луча №1
{
    i=1;
    i=i+1;      //т.к. переменные строковые, то сумма будет равна «11»
    j=1;
    j=str(j+1);             // str - это функция преобразования числа к строке. Внутри функции
 str вначале происходит конвертация всех строковых переменных (в случае их наличия) в
целочисленные, затем происходит сложение чисел, следовательно сумма будет равна «2»
}
```

```
OnEvent("GRAY",N,"ON")  // Выполнится при замыкании любого луча
{
    if(strequal(N,"3")
    {
       // выполнится если это луч 3
    }
}
```

8.2.2.2 Создание собственных процедур

Все собственные процедуры, описанные в скрипте, должны находиться в том же теле программы и перед процедурами, в которых они вызываются.

```
procedure ProcedureName(список параметров){
    //тело процедуры
}
```

> **Внимание!**
>
> Имена параметров должны состоять из одного символа в верхнем регистре.

Примеры использования:

```
procedure ProcedureName(A,B)

{
    n=A+" "+B;
    //при запуске макроса 1 n=«Макрокоманда 1», при запуске макроса 16 n=«Макрокоманда 16»
}
```

```
OnEvent("MACRO",N,"RUN")

{
    a1=N;
    a2="Макрокоманда";
    ProcedureName(a2,a1);
}
```

8.2.3 Описание операторов

Список операторов, используемых для описания действий:

1. **DoReact**(тип объекта, номер, действие[,параметры]) – выполнить действие.
   Пример использования:

```
OnEvent("GRAY","1","ON")
{
                DoReact("GRELE","1","ON");      //при замыкании луча  1  замкнуть реле
 1
}
```

2. **DoCommand**(командная строка) – запуск командной строки.

   Пример использования:

```
OnEvent("GRAY","1","ON")
{
                DoCommand("notepad.exe"); //при замыкании луча  1  запустить
 «Блокнот»
}
```

3. **Wait**(кол-во секунд) – ждать N секунд;
   Sleep(кол-во миллисекунд) – ждать N миллисекунд.

   Операторы ожидания должны быть выделены в отдельный поток. Отдельный поток выделяется квадратными скобками.

   Пример**.** При замыкании Луча №1 Реле №1 будет замыкаться на 5 секунд.

```
OnEvent("GRAY","1","ON")
{
                [
                DoReact("GRELE","1","ON");
                Wait(5);
                DoReact("GRELE","1","OFF");
                ]
}
```

4. Функция проверки состояния объекта:

   CheckState(тип объекта, номер, состояние) – результат будет равен 1, если состояние объекта соответствует действительности, иначе 0.

   В качестве параметров могут быть выражения. Константные значения берутся в кавычки.

   Пример. При замыкании луча №1 проверяется состояние камеры №2 и если состояние «Тревога», то замкнуть реле №1.

```
OnEvent("GRAY","1","ON")
{
                if(CheckState("CAM","2","ALARMED"))
                {
                                DoReact("GRELE","1","ON");
                }
}
```

5. **Условный оператор**:

```
If (выражение)                {
                ... // если результат выражения не 0
                }
                else
                {
                ... // если результат выражения равен 0
                }
```

   Часть оператора else {} может отсутствовать.

   Пример использования:

```
OnEvent ("MACRO","1","RUN"){
                x=5;
                if(x>10) {y=2;}    // если "x" больше чем 10, то y=2
                else {y=3;}           //иначе y=3
}
```

6. **Оператор цикла**:

```
For (выражение 1; выражение 2; выражение 3){
                ...
}
```

   Выражение 1 выполнится в начале цикла, пока выражение 2 истинно, будет выполняться тело цикла, после каждого выполнения тела цикла будет выполняться выражение 3.

   Пример. При замыкании луча №1 реле №1 будет замыкаться и размыкаться с интервалом в 1 секунду и это будет происходить 10 раз.

```
OnEvent ("GRAY","1","ON")
{
                [
                for(i=0;i<10;i=str(i+1))
                {
                  DoReact("GRELE","1","ON");
                  Wait(1);
                  DoReact("GRELE","1","OFF");
                  Wait(1);
                }
                ]
}
```

7. **DoReactGlobal**(тип объекта, номер, состояние) – функция, генерирующая реакции системных объектов. При этом генерируемая реакция рассылается по всем ядрам системы, соединенным по сети.

   Пример. При выполнении макрокоманды №1 ставить камеру №1 на охрану.

```
OnEvent("MACRO","1","RUN")
{
    DoReactGlobal("CAM","1","ARM");
}
```

8. **NotifyEventGlobal**(тип объекта, номер, состояние) – функция, генерирующая системные события. При этом генерируемые события рассылаются по всем ядрам системы, соединенным по сети.

   Пример. При выполнении макрокоманды №1 генерировать событие «Запись на диск» для камеры №1. Команду отправлять по всем ядрам системы в виде события для регистрации в Протоколе событий.

```
OnEvent("MACRO","1","RUN")
{
    NotifyEventGlobal ("CAM","1","REC");
}
```

> **Примечание.**
>
> Если нет необходимости в рассылке события по всем ядрам системы, можно воспользоваться функцией **NotifyEvent**.

8.2.4 Операции и выражения

В таблице представлены общее описание и примеры использования операций сравнения, арифметических и условных операций.

Оператор	**Общее описание, пример использования**
Операции сравнения	
>	Оператор сравнения − больше. Пример см. в разделе Описание операторов
<	Оператор сравнения − меньше. Пример см. в разделе Описание операторов
Арифметические операции	

+	Операция сложение. Пример использования:
	`OnEvent ("MACRO","1","RUN")`
	`{`
	`  x=5;`
	`  y=10;`
	`  i=x+y;  // складывает как строковые т.е. 5+10=510`
	`  e=str(x+y); // складывает как числа 5+10=15`
	`}`

-	Операция вычитание. Пример использования:
	`OnEvent ("MACRO","1","RUN")`
	`{`
	`  x=5;`
	`  y=10;`
	`  i=x-y;  //  вычитание как числа 5-10=-5`
	`  e=str(x-y); // вычитание как числа 5-10=-5`
	`}`

Оператор	Общее описание, пример использования

*	Умножение. Пример использования:
	`OnEvent ("MACRO","1","RUN")`
	`{`
	`  x=5;`
	`  y=10;`
	`  i=x*y;  // умножает как числа 5*10=50`
	`  e=str(x*y); // умножает как числа 5*10=50`
	`}`

/	Деление. Пример использования:
	`OnEvent ("MACRO","1","RUN")`
	`{`
	`  x=5;`
	`  y=10;`
	`  i=x/y;  // делит как числа 5/10=0.5`
	`  e=str(x/y); // делит как числа 5/10=0.5`
	`}`

%	Остаток от целочисленного деления. Пример использования.
	`OnEvent ("MACRO","1","RUN")`
	`{`
	`  a=1120.0;`
	`  b=100;`
	`  e=a%b;  // остаток от целочисленного деления, т.е. 1100 делится на 100, а 20 – это остаток.`
	`  // если делится без остатка то результат = 0`
	`}`

( )	Группа арифметических операций. Пример использования.
	`OnEvent ("MACRO","1","RUN")`
	`{`
	`  x=100/((5*8)/1.028);`
	`}`

Логические операции	

Оператор	Общее описание, пример использования

&&	Оператор логическое И. Пример использования:
	`OnEvent ("MACRO","1","RUN")`
	`{`
	`a=1;`
	`b=2;`
	`z=3;`
	`if((a<b)&&(b<z))`
	`{`
	`y=1; //если ложь, то else`
	`}`
	`else {x=0;}`
	`}`

!	Оператор логического отрицания. Пример использования:
	`OnEvent ("CAM",N,"MD_START")`
	`{`
	`if(!(strequal(N,"1",)))`
	`{`
	`DoReact("GRELE","1",""ON)`
	`}`
	`else`
	`{`
	`DoReact("GRELE","2",""ON)`
	`}`
	`}`

8.2.5 Описание функций

Общее описание и примеры использования математических функций, функций преобразования, форматирования и строковых функций показаны в таблице.

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**
МАТЕМАТИЧЕСКИЕ	

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**
sin[1]	Тригонометрическая функция расчета синуса угла. Формат: y=sin(x); где y – значение функции, x – аргумент функции (в радианах) Пример: y=sin(1.6) Полученное событие: Event : CORE VAR_CHANGED nt_obj_id<1>,value<0.997495>,name<y>,time<15:26:41>,date<21-09-04>
cos[1]	Тригонометрическая функция расчета косинуса угла. Формат: y=cos(x); где y – значение функции, x – аргумент функции (в радианах) Пример: y=cos(2.2) Полученное событие: Event : CORE VAR_CHANGED int_obj_id<1>,value<-0.588501>,name<y>,time<16:00:45>,date<21-09-04>
tan[1]	Тригонометрическая функция, возвращает тангенс угла. Формат: y=tan(x); где y – значение функции, x – аргумент функции (в радианах) Пример: y=tan(1) Полученное событие: Event : CORE VAR_CHANGED int_obj_id<1>,value<1.557408>,name<y>,time<16:43:45>,date<21-09-04>

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**
asin[1]	Возвращает арксинус заданного числового выражения. Формат: y=asin(x); где y – значение функции (в радианах), x – аргумент Пример: y=asin(0.5) Полученное событие: Event : CORE VAR_CHANGED int_obj_id<1>,value<0.523599>,name<y>,time<16:46:39>,date<21-09-04>
acos[1]	Возвращает арккосинус заданного числового выражения. Формат: y=acos(x); где y – значение функции (в радианах), x – аргумент Пример: y=acos(0.55) Полученное событие: Event : CORE VAR_CHANGED int_obj_id<1>,value<0.988432>,name<y>,time<16:46:39>,date<21-09-04>
atan[1]	Возвращает арктангенс заданного числового выражения. Формат: y=atan(x); где: y – значение функции (в радианах), x – аргумент Пример: y=atan(1.2) Полученное событие: Event : Event : CORE VAR_CHANGED int_obj_id<1>,value<0.876058>,name<y>,time<17:07:09>,date<21-09-04>

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**
sinh[1]	Функция sinh возвращает гиперболический синус значения аргумента. Формат: y=sinh(x); где y – значение функции, x – аргумент функции Пример: y=sinh(0.8) Полученное событие: Event : CORE VAR_CHANGED int_obj_id<1>,value<0.888106>,name<y>,time<17:12:26>,date<21-09-04>
cosh[1]	Функция cosh возвращает гиперболический косинус значения аргумента. Формат: y=cosh(x); где y – значение функции, x – аргумент функции Пример: y=cosh(0.35) Полученное событие: Event : CORE VAR_CHANGED int_obj_id<1>,value<0.336376>,name<y>,time<17:25:25>,date<21-09-04>
tanh[1]	Тригонометрическая функция расчета угла. Формат: y=tanh(x); где y – значение функции, x – аргумент функции Пример: y=tanh(0.35) Полученное событие: Event : CORE VAR_CHANGED int_obj_id<1>,value<1.419068>,name<y>,time<17:25:25>,date<21-09-04>

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**
exp[1]	Возвращает значение функции e^x, где x - заданное числовое выражение. Формат: y=exp(x); где y – значение функции, x – аргумент Пример: y=exp(1.65) Полученное событие: Event : CORE VAR_CHANGED int_obj_id<1>,value<5.20698>,name<y>,time<17:39:22>,date<21-09-04>
log[1]	Возвращает натуральный логарифм (по основанию «e») заданного числового выражения. Формат: y=log(x); где: y – значение функции, x – аргумент Пример: y=log(0.65) Полученное событие: Event : CORE VAR_CHANGED int_obj_id<1>,value<-0.430783>,name<y>,time<17:43:22>,date<21-09-04>
log10[1]	Возвращает десятичный логарифм (по основанию 10) заданного числового выражения. Формат: y=log10(x); где y – значение функции, x – аргумент Пример: y=log10(0.05) Полученное событие: Event : CORE VAR_CHANGED int_obj_id<1>,value<-1.30103>,name<y>,time<17:46:28>,date<21-09-04>

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**
sqrt[1]	Возвращает квадратный корень из заданного числового выражения. Формат: y=sqrt(x); где y – значение функции, x – аргумент Пример: y=sqrt(9) Полученное событие: Event : CORE VAR_CHANGED int_obj_id<1>,value<3>,name<y>,time<17:25:25>,date<21-09-04>
abs[1]	Функция abs возвращает абсолютное значение целого аргумента. Формат: y=abs(x); где y – значение функции, x – аргумент Пример: y= abs(-1) Полученное событие: Event : CORE VAR_CHANGED int_obj_id<1>,value<1>,name<y>,time<13:39:37>,date<22-09-04>
deg[1]	Тригонометрическая функция расчета угла. Возвращает градусную меру. Формат: y=deg(x); где y – значение функции в градусах, x – значение аргумента в радианах Пример: y=deg(3.14) Полученное событие: Event : CORE VAR_CHANGED int_obj_id<1>,value<179.908748>,name<y>,time<13:13:51>,date<22-09-04>

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**
rad[1]	Тригонометрическая функция расчета угла. Формат: y=rad(x); где y – значение функции в радианах, x – значение аргумента в градусах Пример: y=rad(180) Полученное событие: Event : CORE VAR_CHANGED value<3.141593>,name<y>,time<15:04:17>,date<17-03-08>
ПРЕОБРАЗОВАНИЕ	
floor[1]	Функция преобразования до целого числа в меньшую сторону. Формат: x= floor(y); где x – значение функции, y – дробное или целое число Пример: x= floor(5.55) Полученное событие: Event : CORE VAR_CHANGED int_obj_id<1>,value<5>,name<x>,time<20:51:48>,date<21-09-04>
ceil[1]	Функция преобразования до целого числа в большую сторону. Формат: x= ceil (y); где x – значение функции, y – дробное или целое число Пример: x= ceil(5.55) Полученное событие: Event : CORE VAR_CHANGED int_obj_id<1>,value<6>,name<x>,time<20:51:48>,date<21-09-04>

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**

str[1]	Функция преобразования числа к строке.
	Формат: x=str(y); где x – значение функции, y – аргумент
	Пример:
	z=(9);
	a=str(z);
	b=sqrt(a);
	Полученные события:
	Event : CORE VAR_CHANGED int_obj_id<1>,value<9>,name<z>,time<14:27:31>,date<22-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<9>,name<a>,time<14:27:31>,date<22-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<3>,name<b>,time<14:27:31>,date<22-09-04>

atof[1]	Функция преобразования строки в число. Формат: x=atof(y); где x – значение функции, y – аргумент Пример: x="0"; x=str(atof(x)+10); Полученное событие: Event : CORE VAR_CHANGED value<0>,name<x>,time<15:34:44>,date<17-03-08> Event : CORE VAR_CHANGED value<10>,name<x>,time<15:34:44>,date<17-03-08>

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**
val[1]	Функция преобразования строки в число. Формат: x=val(y); где x – значение функции, y – аргумент Пример: x="10"; x=str(val(x)+2); Полученное событие: Event : CORE VAR_CHANGED value<10>,name<x>,time<15:34:44>,date<17-03-08> Event : CORE VAR_CHANGED value<12>,name<x>,time<15:34:44>,date<17-03-08>

int[1]	Преобразование дробного числа в целое (отбросить дробную часть).
	Формат: x=int(y); где x – значение функции, y – аргумент (дробное число для преобразования)
	Пример:
	y=(2.33);
	x=int(y);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value<2.33>,name<y>,time<16:05:28>,date<22-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<2>,name<x>,time<16:05:28>,date<22-09-04>

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**
long2time[1]	Используется для преобразования заданного количества секунд во время. Формат: x=long2time(y); где x – значение функции(время), y – число в секундах Формат исходной записи (аргумента): <ММ> Формат полученной записи: <ЧЧ:ММ:СС> Пример: x=long2time(12345); Полученное событие: Event : CORE VAR_CHANGED int_obj_id<1>,value<03:25:45>,name<x>,time<13:53:02>,date<20-09-04>.

time2long[1]	Преобразовать время в количество секунд.
	Формат: x=time2 long(y); где x – значение в секундах, y – время в формате <часы>.<минуты>
	Пример:
	y=(0.15);
	x=time2long(y);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value<0.15>,name<y>,time<19:39:49>,date<22-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<900>,name<x>,time<19:39:49>,date<22-09-04>

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**

scalar2date[1]	Преобразовать количество дней в дату (количество дней исчисляется с начала нашей эры).
	Формат: x=scalar2date (y); где x – значение(дата), y – количество дней
	Пример:
	y=(731500);
	x=scalar2date(y);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value<731500>,name<y>,time<19:57:46>,date<22-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<12-10-03>,name<x>,time<19:57:46>,date<22-09-04>

scalar[1]	Преобразовать дату в количество дней (количество дней исчисляется с начала нашей эры). Формат: x=scalar(y); где x – числовое значение (в сутках), y – дата Формат записи: <ДД.ММ.ГГГГ> Пример: x=scalar(“19.10.2004”) Полученное событие: Event : CORE VAR_CHANGED int_obj_id<10>,value<731873>,owner<WS1>,name<x>,time<15:24:11>, guid_pk<{42E93AF5-4862-485E-AEF6-D14C7BF79C5B}>,date<08-12-09>

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**

convert_num[1]	Преобразовать число в строку написания этого числа.
	Формат: x=convert_num(y); где x – строковое значение числа, y – преобразуемое число
	Пример:
	y=(24009921);
	x=convert_num(y);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value<24009921>,name<y>,time<12:37:20>,date<23-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<Двадцать четыре миллиона девять тысяч девятсот двадцать один >,name<x>,time<12:37:20>,date<23-09-04>

convert_cur[1]	Преобразовать число (денежную сумму) в строку написания этого числа и добавить руб. и коп.
	Формат: x=convert_cur(y); где x – строковое значение денежной суммы, y – число(денежная сумма)
	Формат записи: <PP.KK>
	Пример:
	y=(17999.98);
	x=convert_cur(y);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value<17999.98>,name<y>,time<12:49:30>,date<23-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<Семнадцать тысяч девятсот девяносто девять рублей 98 копеек>,name<x>,time<12:49:30>,date<23-09-04>

ФОРМАТИРОВАНИЯ	

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**

number_frm[2]	Форматирование числа.
	Формат: x=number_frm(y,z); где x – значение функции, y-исходное число, z – количество цифр после запятой
	Пример:
	y=(17999.09998);
	x=number_frm(y,3);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value<17999.09998>,name<y>,time<14:21:24>,date<23-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<17999.100>,name<x>,time<14:21:24>,date<23-09-04>

int_frm[2]	Форматирование числа.
	Формат: x=int_frm(y,z); где x – значение, y-исходное число, z – количество цифр в числе на выходе
	Пример:
	y=(17999.99);
	x=int_frm(y,10);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value<17999.99>,name<y>,time<14:31:46>,date<23-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<0000017999>,name<x>,time<14:31:46>,date<23-09-04>

currency_std[1]	Форматирование значения числа, представляющего деньги (замена '.' на '-'). Формат: x=currency_std(y); где x – значение функции с измененным форматом, y – число (денежная сумма) Пример: x=currency_std(3.62); Полученное событие: Event : CORE VAR_CHANGED int_obj_id<1>,value<3-62>,name<x>,time<13:40:01>,date<23-09-04>

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**
IsVarExist[1]	Функция проверки заданного параметра в событии. Формат: y=IsVarExist(“x”); где y - значение, x – параметр Если параметр существует, возвращается «1», иначе «0». Пример: p=IsVarExist(“param0”) Полученное событие: Event : CORE VAR_CHANGED int_obj_id<10>,value<0>,owner<WS1>,name<p>,time<12:02:11>, guid_pk<{6A8B5BC9-919C-4098-844A-FBF78FA20820}>,date<14-12-09>
GetObjectIdByParam [3]	Функция, возвращающая первый найденный идентификатор объекта по заданному значению параметра. Id=GetObjectIdByParam ("x","y","z"); где id - возвращаемое значение, x – тип объекта, y – параметр, z – значение параметра Пример: Id=GetObjectIdByParam("CAM","color","0"); Полученное событие: Event : CORE VAR_CHANGED date<28-02-11>,value<2>,int_obj_id<1>,fraction<218>,name<Id>, guid_pk<{F903A28C-3243-E011-901F-6CF049E58698}>,time<15:02:04>,owner<D-IVANOV> * Id=2 (см. value<2>), если функция возвращает пустое значение (value< >), необходимо проверить правильность написания параметров и самой функции.
СТРОКОВЫЕ	

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**

strequal[2]	Сравнение строк.
	Формат: x=strequal(z,y); где x – значение, z и y – сравниваемые строки
	Пример:
	z=str(1019);
	y=str(1019);
	x=strequal(z,y);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value<1019>,name<z>,time<16:51:45>,date<23-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<1019>,name<y>,time<16:51:45>,date<23-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<1>,name<x>,time<16:51:45>,date<23-09-04>
	* «value<1>» (см. пример выше) – в полученном событии мы получаем либо «value<>» – это означает что сравниваемые строки не совпадают, либо «value<1>» – это значит что сравниваемые строки полностью идентичны друг другу.

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**

strsub[2]	Определение наличия подстроки в строке.
	Формат: x=strsub(y,z); где x-значение, y – строка, в которой ведется поиск, z-подстрока
	Пример №1:
	z=str(888123);
	y=str(123);
	x=strsub(z,y);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value<888123>,name<z>,time<16:07:07>,date<23-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<123>,name<y>,time<16:07:07>,date<23-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<4>,name<x>,time<16:04:34>,date<23-09-04>
	Пример №2:
	z="67hb8vc56";
	y="vc";
	x=strsub(z,y);
	Полученное событие:
	Event : CORE VAR_CHANGED value<67hb8vc56>,name<z>,time<12:15:09>,date<18-03-08>
	Event : CORE VAR_CHANGED value<vc>,name<y>,time<12:15:09>,date<18-03-08>
	Event : CORE VAR_CHANGED value<6>,name<x>,time<12:15:09>,date<18-03-08>
	* "value<4>" (см. пример № 1) означает индекс в исходной строке, начиная с которого обнаружено первое вхождение подстроки в эту строку. При отрицательном результате поиска функция возвращает значение value<>.

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**
strempty[1]	Определение пуста ли строка. Формат: x=strempty(y); где x – значение (равно 1 если строка пуста), y – строка Пример: y=""; x=strempty(y); Полученное событие: Event : CORE VAR_CHANGED value< >, name<y>,time<12:27:32>,date<18-03-08> Event : CORE VAR_CHANGED value<1>,name<x>,time<12:27:32>,date<18-03-08> * значение функции value <> означает, что строка не пуста.

straleft[2]	Выравнивание влево.
	Формат: x=straleft(y,z); где x – выровненная строка, y – строка, z – значение выравнивания.
	Пример:
	y=str(123456789);
	x=straleft(y,5);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value<123456789>,name<y>,time<18:04:05>, date<23-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<12345>,name<x>,time<18:04:05>,date <23-09-04>
	Примечание. В случае, если число z больше, чем количество символов в строке, функция дополняет исходную строку пробелами справа до тех пор, пока ее длина не станет равна числу z.

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**

strmid[3]	Взять подстроку.
	Формат: x=strmid(y,z,w); где x – строковое значение, y – строка, z – с какой позиции строки, w – длина подстроки
	Пример:
	z=(7);//с какой позиции
	w=(9);//длина
	x=strmid("взять подстроку (1 - строка, 2 - с какой позиции, 3 - длинна)",z,w);
	y=strmid("взять подстроку (1 - строка, 2 - с какой позиции, 3 - длинна)",17,10);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value<6>,name<z>,time<14:18:08>,date<24-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<9>,name<w>,time<14:18:08>,date<24-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<подстроку>,name<x>,time<14:18:08>,date<24-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<1 – строка>,name<y>,time<14:18:08>,date<24-09-04>

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**

strletf[2]	Взять левую часть строки.
	Формат: y=strleft(s,w); где y –строковое значение, s – строка, w – длина (с начала строки)
	Пример:
	w=(5);//длина
	s=("Взять левую часть строки");//строка
	y=strleft(s,w);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value<5>,name<w>,time<14:54:31>,date<24-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<Взять левую часть строки>,name<s>,time<14:54:31>,date<24-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<Взять>,name<y>,time<14:54:31>,date<24-09-04>

strright[2]	Взять правую часть строки (1 – строка, 2 – длина).
	Формат: y=strleft(s,w); где y – строковое значение, s – строка, w – длина (с конца строки)
	Пример:
	w=(6);//длина
	s=("Взять правую часть строки");//строка
	y=strright(s,w);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value<6>,name<w>,time<15:10:36>,date<24-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<Взять правую часть строки>,name<s>,time<15:10:36>,date<24-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<строки>,name<y>,time<15:10:36>,date<24-09-04>

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**

strnleft[2]	Взять без левой части строки.
	Формат: y=strnleft(s,w); где y – строковое значение, s – строка, w – длина левой части, которая будет отсечена
	Пример:
	w=(6);//длина
	s=("взять без левой части строки");//строка
	y=strnleft(s,w);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value<6>,name<w>,time<15:32:38>,date<24-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<взять без левой части строки>,name<s>,time<15:32:38>,date<24-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<без левой части строки>,name<y>,time<15:32:38>,date<24-09-04>

srtnright[2]	Взять без правой части строки.
	Формат: y=strnright(s,w); где y – строковое значение, s – строка, w – длина правой части, которая будет отсечена.
	Пример:
	w=(6);//длина
	s=("взять без правой части строки");//строка
	y=strnright(s,w);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value<6>,name<w>,time<15:44:31>,date<24-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<взять без правой части строки>,name<s>,time<15:44:31>,date<24-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<взять без правой части >,name<y>,time<15:44:31>,date<24-09-04>

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**

get_substr[3]	Взять подстроку (1 – строка, 2 – подстрока с которой начать, 3 – подстрока которой завершить, "\r" – конец строки).
	Формат: y=get_substr(s,w,x); где y – значение(подстрока), s – строка, w – подстрока с которой начать, x – подстрока которой завершить("\r" – конец строки)
	Формат записи: <NN.NN>
	Пример:
	s=("взять подстроку 1234567890");//строка
	w=("по");//подстрока с которой начать
	x=("\r");//подстрока которой завершить, "\r" – конец строки
	y=get_substr(s,w,x);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value<взять подстроку 1234567890>,name<s>,time<16:34:13>,date<24-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<по>,name<w>,time<16:34:13>,date<24-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<\r>,name<x>,time<16:34:13>,date<24-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<подстроку 1234567890>,name<y>,time<16:34:13>,date<24-09-04>
	Пример:
	s=("взять подстроку 1234567890");//строка
	w=("по");//подстрока с которой начать
	x=(1);//подстрока которой завершить, "\r" – конец строки
	y=get_substr(s,w,x);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value<взять подстроку 1234567890>,name<s>,time<16:36:26>,date<24-09-04>

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**
	Event : CORE VAR_CHANGED int_obj_id<1>,value<по>,name<w>,time<16:36:26>,date<24-09-04> Event : CORE VAR_CHANGED int_obj_id<1>,value<1>,name<x>,time<16:36:26>,date<24-09-04> Event : CORE VAR_CHANGED int_obj_id<1>,value<подстроку >,name<y>,time<16:36:26>,date<24-09-04>

strltrim[1]	Убрать пробелы слева.
	Формат: y=strltrim(w); где y – полученное строковое значение, w – строка
	Пример:
	w=(" убрать пробелы слева");//строка
	y=strltrim(w);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value< убрать пробелы слева>,name<w>,time<17:07:49>,date<24-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<убрать пробелы слева>,name<y>,time<17:07:49>,date<24-09-04>

strrtrim[1]	Убрать пробелы справа.
	Формат: y=strrtrim(w); где y – полученное строковое значение, w – строка
	Пример:
	w=("Убрать пробелы справа ");//строка
	y=strrtrim(w);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value<Убрать пробелы справа >,name<w>,time<17:18:35>,date<24-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<Убрать пробелы справа>,name<y>,time<17:18:35>,date<24-09-04>

Функции **(В квадратных скобках указано количество исполняемых параметров)**	**Общее описание, пример использования**

stratrim[1]	Убрать пробелы с обеих сторон.
	Формат: y=stratrim(w); где y – полученное строковое значение, w – строка
	Пример:
	w=(" убрать пробелы с обеих сторон ");//строка
	y=stratrim(w);
	Полученное событие:
	Event : CORE VAR_CHANGED int_obj_id<1>,value< убрать пробелы с обеих сторон >,name<w>,time<17:27:44>,date<24-09-04>
	Event : CORE VAR_CHANGED int_obj_id<1>,value<убрать пробелы с обеих сторон>,name<y>,time<17:27:44>,date<24-09-04>

Примечание.

Функции date<ДД-ММ-ГГ> и time<ЧЧ:ММ:СС> возвращают текущие дату и врем я соответственно. Функция pi<3,14159265358979323846264338327 95> возвращает значение числа π.

8.3 Примеры скриптов на встроенном языке

8.3.1 Примеры с Камерами и Монитором видеонаблюдения

GRABBER Устройство видеоввода

MACRO Макрокоманда

CAM Камера

MONITOR Монитор видеонаблюдения

8.3.1.1 Форматы и функции

Формат процедуры событий для **Устройства видеоввода**:

```
OnEvent("GRABBER","_id_","_событие_")
```

Формат оператора для описания действий с **Устройством видеоввода**:

```
DoReact("GRABBER","_id_","_команда_" [,"_параметры_"]);
```

Формат процедуры событий для объекта **Камера**:

```
OnEvent("CAM","_id_","_событие_")
```

Формат оператора для описания действий с **Камерой**:

```
DoReact("CAM","_id_","_команда_" [,"_параметры_"]);
```

Функция проверки состояния объекта **Камера**:

```
CheckState("CAM","номер","состояние")
```

Формат процедуры событий для объекта **Монитор**:

```
OnEvent("MONITOR","_id_","_событие_")
```

Формат оператора для действий с **Монитором**:

```
DoReact("MONITOR","_id_","_команда_"[,"_параметры_"]);
```

8.3.1.2 Примеры

Примеры использования событий и реакций объекта **Устройство видеоввода**:

1. Необходимо установить для первого устройства видеоввода первый канал, максимальную скорость оцифровки, разрешение полукадр и формат PAL при запуске первой макрокоманды.

```
OnEvent("MACRO","1","RUN") //запуск макрокоманды 1
{
```

```
    DoReact("GRABBER","1", "SETUP", "chan<1>,mode<0>,resolution<1>,format<PAL>")
;
    //установка для первой платы видеоввода канал - 1, скорость оцифровки -
максимальная, разрешение - полукадр, формат - PAL
}
```

2. Необходимо при запуске третьей макрокоманды установить диски D:\ и F:\ для записи видеоархива.

```
OnEvent("MACRO","3","RUN") //запуск макрокоманды 3
{
    DoReact("GRABBER","1","SET_DRIVES","drives<D:\,F:\>"); //запись видеоархива на
диски D:\ и F:\
}
```

3. Необходимо вывести первую видеокамеру на первый аналоговый выход платы и отключить первые аналоговые выходы первой и второй плат при ошибке подключения ко второй плате видеоввода.

```
OnEvent("GRABBER","2"," UPS_FATAL_ERROR") //ошибка подключения к плате видеоввода 2
{
    DoReact("CAM","1","MUX1"); //вывод видеокамеры 1 на 1-ый аналоговый вывод платы
    Wait(5);
    DoReact("GRABBER","1","MUX1_OFF"); //отключение 1-го аналогового выхода первой
платы
    DoReact("GRABBER","2","MUX1_OFF"); //отключение 1-го аналогового выхода второй
платы
}
```

> **Примечание.**
>
> Если аналоговые выходы двух и более плат соединяются параллельно, и видеокамера 1, например, принадлежит первому грабберу, а видеокамера 2 – второму, то при вызове команды «DoReact("CAM","1","MUX1");» необходимо сначала вызвать команду «DoReact("GRABBER","2","MUX1_OFF");» и, соответственно, при вызове команды «DoReact("CAM","2","MUX1");» необходимо сначала вызвать команду «DoReact("GRABBER","1","MUX1_OFF");». Иначе произойдет наслоение сигналов.

4. Необходимо отключить второй аналоговый выход платы видеоввода при восстановлении питания от сети.

```
OnEvent("GRABBER","1","UPS_ONLINE") //восстановление питания от сети
{
```

```
    DoReact("GRABBER","1","MUX2_OFF"); //отключение аналогового выхода 2
}
```

Примеры использования событий и реакций объекта **Камера**:

1. При постановке первой камеры на охрану выполнить перевод камеры в цветной режим и начать запись с нее.

```
OnEvent("CAM","1","ARM") //первая видеокамера поставлена на охрану
{
    DoReact("CAM","1","SETUP","color<1>"); //установка цветного режима видеокамеры
    DoReact("CAM","1","REC"); //запись с первой видеокамеры
}
```

2. Необходимо поставить на охрану первую видеокамеру при отключении пятой видеокамеры.

```
OnEvent("CAM","5","DETACH") //пятая видеокамера отключена
{
    DoReact("CAM","1","ARM"); //первая видеокамера поставлена на охрану
}
```

3. Необходимо использовать половину ресурсов при записи у первой видеокамеры (то есть, если в системе через первую плату видеоввода подключено 4 видеокамеры, то первая будет записывать со скоростью 6 кадров/сек, а остальные три – по 2-2,5 кадра/сек.), если она находится в тревожном состоянии.

```
OnEvent("CAM","1","MD_START") //первая видеокамера находится в тревожном состоянии
{
    DoReact("CAM","1","SETUP","rec_priority<2>"); //использование половины ресурсов
при записи
}
```

4. Необходимо установить максимальную компрессию синхронно с четвертым микрофоном звуковой платы на первой видеокамере при записи на диск видео с первой видеокамеры.

```
OnEvent("CAM","1","REC") //первая видеокамера ведет запись на диск
{
    DoReact("CAM", "1", "SETUP", "compression<5>, audio_type<OLXA_LINE>,
audio_id<4>"); //первая видеокамера, максимальная компрессия,
синхронно с четвертым микрофоном звуковой платы
}
```

5. Необходимо начать запись с первой видеокамеры с минимальным качеством в черно-белом режиме, когда она выйдет из состояния тревоги.

```
OnEvent("CAM","1","MD_STOP") //первая видеокамера перестала находиться в тревожном состоянии
{
    value = 5;
    DoReact("CAM", "1", "SETUP", "compression<" + value + ">,color<0>");
    //начать запись первой видеокамеры с минимальным качеством в ч/б режиме
}
```

6. Необходимо начать запись с первой видеокамеры в режиме «откат», когда она снята с охраны.

```
OnEvent("CAM","1","DISARM")  //первая видеокамера снята с охраны
{
    DoReact("CAM","1","REC","rollback<1>");  //начать запись с первой видеокамеры в режиме «откат»
}
```

7. Установить новые параметры видеоканала при подключении первой видеокамеры.

```
OnEvent("CAM","1","ATTACH") //подключена первая видеокамера
{
    VIDEO_CANAL_ID = GETOBJECTPARAM("CAM","1","PARENT_ID"); //определяем идентификатор видеоканала, которому принадлежит первая видеокамера
    DoReact("GRABBER",VIDEO_CANAL_ID,"SETUP","chan<0>,mode<0>,resolution<1>,format<pal>"); //устанавливаем новые параметры видеоканала
}
```

8. По макрокоманде 2 запустить автопанорамирование на камере 1.

```
OnEvent ("MACRO","2","RUN")
{
    DoReact("CAM","1","CRUISE_START","cruise_id<1>,action<CRUISE_START>,cam_id<1>");
}
```

9. Есть определенное количество камер (num). Необходимо проверить работу детектора движения по всем камерам (можно использовать для проверки работоспособности датчиков охраны).

   Для решения задачи используется эмуляция линейного символьного массива (строка), т.е. заполняется массив символов (в примере это символ «N»). Далее при сработке детектора движения по камере меняется соответствующий (идентификатору камеры) элемент массива (меняется на "Y"). Таким образом, на выходе образуется символьный массив из «N» (камера не сработала) и «Y» (камера сработала). Подсчитывается количество сработок и выдается сообщение об общем количестве камер и количество камер, у которых сработал детектор. Старт проверки по Макрокоманде 1. Остановка по Макрокоманде 2.

```
OnInit()
{
    run=0;
}

OnEvent("MACRO","1","RUN")
{
    run=1; flag=""; num=8;
    for(i=1;i<str(num+1);i=str(i+1))
    {
        DoReact("CAM",i,"DISARM");
        DoReact("CAM",i,"REC_STOP");
        DoReact("CAM",i,"ARM");
        flag=flag+"N";
        if(i<num) {flag=flag+"|";}
    }
}

OnEvent("CAM",N,"MD_START")
{
    if(run)
    {
        nn=str((N*2)-1);
        flag=strleft(flag,str(nn-1))+"Y"+strright(flag,str(((num*2)-1)-nn));
    }
}

OnEvent("MACRO","2","RUN")
{
    run=0; fin=0;
    for(i=1;i<str(num+1);i=str(i+1))
    {
        tmp=extract_substr(flag,"|",str(i-1));
        if(strequal(tmp,"Y")) {fin=str(fin+1);}
        DoReact("CAM",i,"DISARM");
    }
    tmp="Всего:"+str(num)+" Сработало:"+str(fin);
    rez=MessageBox("",tmp,0);
}
```

10. При возникновении тревоги по камере 1 накладывать титры на видеоизображение с данной камеры. При окончании тревоги накладывать титры об окончании тревоги.

```
OnEvent("CAM","1","MD_START")
{
    DoReact("CAM","1","CLEAR_SUBTITLES","title_id<1>"); //удалить все титры с видеоизображения
    DoReact("CAM","1","ADD_SUBTITLES","command<Камера 1 Тревога " + time + "\r>,page<BEGIN>,title_id<1>");
```

```
    //параметр time позволяет включить в титры время регистрации события
}

OnEvent("CAM","1","MD_STOP")
{
    DoReact("CAM","1","ADD_SUBTITLES","command<Камера 1 Конец тревоги " + time +
"\r>,page<END>,title_id<1>");
}
```

> **Примечание**
>
> При использовании параметров page<BEGIN> и page<END> будут заполняться соответствующие поля в базе титров, что даст возможность производить поиск данных с помощью интерфейсного объекта **Поиск по титрам**.

Примеры использования событий и реакций объекта **Монитор**:

1. Необходимо при запуске первой макрокоманды проиграть запись с видеокамеры 1 на мониторе 4 с указанными датой и временем.

```
OnEvent("MACRO","1","RUN")
{
    DoReact("MONITOR","4","ARCH_FRAME_TIME","cam<1>,date<"+date+">,time<11:00:0
0>");
    DoReact("MONITOR","4","KEY_PRESSED","key<PLAY>");
}
```

2. Необходимо при печати кадра с первой видеокамеры перейти в режим просмотра видеоархива на первой видеокамере монитора 4, и перейти на 10 кадров далее, начиная с фрагмента указанной даты и времени.

```
OnEvent("CAM", "1", "PRINT")
{
    DoReact("MONITOR","4","ARCH_FRAME_TIME","cam<1>,date<"+date+">,time
<11:00:00>");
    for(i=0;i<10;i=i+1)
    {
        DoReact ("MONITOR","4","KEY_PRESSED","key<FF>");
    }
}
```

3. Необходимо приблизить видеоизображение на экране монитора, если видеокамера находится в состоянии тревоги, и вернуть в исходное состояние при ее окончании.

```
OnEvent("CAM","1","MD_START")
{
    DoReact("MONITOR","1","KEY_PRESSED","key<ZOOM_IN>");
}

OnEvent("CAM", "1", "MD_STOP");
{
    DoReact("MONITOR","1","KEY_PRESSED","key<ZOOM_OUT>");
}
```

4. Необходимо вывести на экран монитора раскладку под номером 1 при срабатывании макрокоманды.

```
OnEvent("MACRO","1","RUN")
{
    DoReact("MONITOR","1","KEY_PRESSED","key<SELECT_LAYOUT>,number<1>");
}
```

5. Команда запуска экспорта видео с Камеры 1 в Мониторе 1, начиная с момента времени 24-10-14 17:10:38 и заканчивая 24-10-14 17:10:50, в файл c:\aaa.avi.
   Примеры запуска экспорта тремя способами: через IIDK (порт 900 и порт 1030) и через скрипт.
   a. **IIDK (порт 900)**
      MONITOR|1|START_AVI_EXPORT|start<24-10-14 17:10:38>,finish<24-10-14 17:10:50>,avi_path<c:\aaa.avi>,cam<1>
   b. **IIDK (порт 1030)**
      CORE||DO_REACT|
      source_type<MONITOR>,source_id<1>,action<START_AVI_EXPORT>,params<4>,param0_name<avi_path>,
      param0_val<c:
      \aaa.avi>,param1_name<cam>,param1_val<1>,param2_name<finish>,param2_val<24-10-14 17:10:50>,
      param3_name<start>,param3_val<24-10-14 17:10:38>
   c. **Скрипт** (запуск по Макрокоманде 1)

```
OnEvent("MACRO","1","RUN")
{
    DoReact("CORE","","DO_REACT","source_type<MONITOR>,source_id<1>,acti
on<START_AVI_EXPORT>,params<4>,
    param0_name<avi_path>,param0_val<c:
\aaa.avi>,param1_name<cam>,param1_val<1>,param2_name<finish>,
     param2_val<24-10-14
17:10:50>,param3_name<start>,param3_val<24-10-14 17:10:38");
}
```

6. По макрокоманде 1 включать управление телеметрией при помощи мыши на камере 4, выведенной на монитор 10, по макрокоманде 2 отключать.

```
OnEvent("MACRO","1","RUN")
{
    DoReact("MONITOR","10","CONTROL_TELEMETRY","cam<4>,on<1>");
}

OnEvent("MACRO","2","RUN")
{
    DoReact("MONITOR","10","CONTROL_TELEMETRY","cam<4>,on<0>");
}
```

7. Выводить активную камеру на аналоговый монитор.

```
OnEvent ("MONITOR","1","ACTIVATE_CAM")
{
    DoReact ("CAM",cam,"MUX1");
}
```

8. Выводить тревожную камеру в режим однократера.

```
OnEvent ("CAM",N,"MD_START")
{
    DoReact ("MONITOR","1","ACTIVATE_CAM","cam<"+N+">");
    DoReact ("MONITOR","1","KEY_PRESSED","key<SCREEN.1>");
}
```

9. Тревожный монитор, на котором всегда остается видео от последней тревожной камеры.

```
OnInit()
{
    counter=0;
}

OnEvent("CAM",T,"MD_START")
{
    if(strequal(counter,"0"))
    {
        DoReact("MONITOR","2","REMOVE_ALL");
        DoReact("MONITOR","2","ADD_SHOW","cam<"+T+">");
    }
    counter=str(counter+1);
}


OnEvent("CAM",M,"MD_STOP")
{
    counter=str(counter-1);
    if(strequal(counter,"0"))
```

```
    {
        DoReact("MONITOR","2","ADD_SHOW","cam<"+M+">");
    }
}
```

8.3.2 Примеры с Компьютером и Экраном

SLAVE Компьютер

DISPLAY Экран

8.3.2.1 Форматы

Формат процедуры событий для объекта **Компьютер**:

```
OnEvent("SLAVE","_id_","_событие_")
```

Формат оператора для описания действий с объектом **Компьютер**:

```
DoReact("SLAVE","_id_","_команда_" [,"_параметры_"]);
```

Формат процедуры событий для объекта **Экран**:

```
OnEvent("DISPLAY","_id_","_событие_")
```

Формат оператора для описания действий с **Экраном**:

```
DoReact("DISPLAY","_id_","_команда_" [,"_параметры_"]);
```

8.3.2.2 Примеры

Примеры использования событий и реакций объекта **Компьютер:**

1. При отсутствии диска для записи архива остановить запись с камеры 2.

```
OnEvent("SLAVE","1"," NO_DISC")
{
```

```
    DoReact("CAM","2"," REC_STOP");
}
```

2. По Макрокоманде 1 получить глубину архива по камере 1.

```
OnEvent ("MACRO","1","RUN"){
   DoReact ("SLAVE","WS3","GET_DEPTH","cam<1>");
}
```

В результате в отладочном окне будет отображена следующая строка:

```
Event : SLAVE|WS3|ARCHIVE_DEPTH|
cam<1>,core_global<1>,date<11-07-13>,depth<42>,destination_id<1>,destination_so
urce<PROGRAM>,fraction<970>,guid_pk<{003DFC83-0CEA-E211-A437-0017C401D5C2}
>,owner<WS3>,param0<01:18>,slave_id<WS3>,time<13:30:33>
```

Кроме того, в Протоколе событий будет отображено событие **Глубина архива**, а в поле **Дополнительная информация** будет указана глубина архива в формате Дни:Часы. Данная информация также отображается в отладочном окне в параметре события **param0<>**.

Пример использования событий и реакций объекта **Экран**:

1. При активировании первой временной зоны отобразить первый экран на компьютере CLIENT.

```
OnEvent("TIME_ZONE","1","ACTIVATE")
{
    DoReact("DISPLAY","1","ACTIVATE","macro_slave_id<CLIENT>");
}
```

2. Есть 2 экрана, первый отображает виртуальный монитор с камерами, второй отображает объект Карта с датчиками ОПС Болид. При сработке тревоги по камере показывается Экран 1, при срабатывании тревоги от датчика показывается Экран 2, но только на компьютере CLIENT.

```
OnEvent("CAM",N,"MD_START")
{
    DoReact("DISPLAY","2","DEACTIVATE","macro_slave_id<CLIENT>");
    DoReact("DISPLAY","1","ACTIVATE","macro_slave_id<CLIENT>");
}

OnEvent("BOLID_ZONE",M,"ALARM")
{
    DoReact("DISPLAY","1","DEACTIVATE","macro_slave_id<CLIENT>");
    DoReact("DISPLAY","2","ACTIVATE","macro_slave_id<CLIENT>");
}
```

8.3.3 Пример с Картой

MAP Карта

Формат процедуры событий для **Карты**:

```
OnEvent("MAP", "_id_","_событие_" [,"_параметры_"])
```

Формат оператора для описания действий с **Картой**:

```
DoReact("MAP","_id_","_команда_" [,"_параметры_"]);
```

Пример. Скрыть Камеру 10 на Карте 1 по Макрокоманде 10.

```
OnEvent("MACRO","10","RUN")
{
    DoReact("MAP","1","HIDE_OBJECT","objtype<CAM>,objid<10>,hide<1>");
}
```

8.3.4 Примеры с Архивом и Внешним хранилищем

ARCH Долговременный архив

IPSTORAGE Внешнее хранилище

8.3.4.1 Форматы

Формат процедуры событий для объекта **Долговременный архив**:

```
OnEvent("ARCH","_id_","_событие_")
```

Формат оператора для описания действий с **Внешним хранилищем**:

```
DoReact("IPSTORAGE","_id_","_команда_" [,"_параметры_"]);
```

8.3.4.2 Примеры

Пример для объекта **Долговременный архив**. Если не производится архивация через Долговременный архив 1, разослать соответствующее событие по всем ядрам системы.

```
OnEvent("ARCH","1","INACTIVE")
{
NotifyEventGlobal ("ARCH","1","INACTIVE");
}
```

Пример для объекта **Внешнее хранилище**. По макрокоманде 10 выполнить импорт архива из внешнего хранилища камеры 45 за период с 11-01-19 16:00:55 по 11-01-19 17:00:55.

```
OnEvent("MACRO","10","RUN")
{
    DoReact("IPSTORAGE", "1", "IMPORT", "cam<45>,datetime_from<11-01-19
16:00:55>,datetime_to<11-01-19 17:00:55>");
}
```

8.3.5 Примеры с Макрокомандами и Временными зонами

MACRO Макрокоманда

TIME_ZONE Временная зона

8.3.5.1 Форматы и функции

Формат процедуры событий для объекта **Макрокоманда**:

```
OnEvent("MACRO","_id_","_событие_")
```

Формат оператора для описания действий с **Макрокомандами**:

```
DoReact("MACRO","_id_","_команда_" [,"_параметры_"]);
```

Функция проверки состояния объекта **Макрокоманда**:

```
CheckState ("MACRO","номер","состояние")
```

Формат процедуры событий для объекта **Временная зона**:

```
OnEvent("TIME_ZONE","_id_","_событие_")
```

Формат оператора для описания действий с **Временной зоной**:

```
DoReact("TIME_ZONE","_id_","_команда_" [,"_параметры_"]);
```

Функция проверки состояния объекта **Временная зона**:

```
CheckState ("TIME_ZONE","номер","состояние")
```

8.3.5.2 Примеры

Примеры использования событий и реакций объекта **Макрокоманда**:

1. Необходимо записать текущее положение видеокамеры в 1 пресет при выполнении макрокоманды 1.

```
OnEvent("MACRO","1","RUN")
{
    DoReact("TELEMETRY","1","SET_PRESET","TEL_PRIOR<1>");
}
```

2. Необходимо выполнить макрокоманду 2, если камера 1 поставлена на охрану.

```
OnEvent("CAM","1","ARM")
{
    DoReact("MACRO", "2", "RUN");
}
```

3. Запускать и останавливать патрулирование поворотного устройства по макрокомандам.

```
OnEvent("MACRO","1","RUN")
{
    DoReact("TELEMETRY","1.1","PATROL_PLAY","tel_prior<1>");
}

OnEvent("MACRO","2","RUN")
{
```

```
    DoReact("TELEMETRY","1.1","STOP","tel_prior<1>");
}
```

4. Пример бесконечного цикла и выхода из него. Старт цикла по макрокоманде 1, остановка по макрокоманде 2.

```
OnEvent("MACRO","1","RUN") //при запуске макрокоманды 1
{
    //квадратные скобки нужны для выделения оператора ожидания в отдельный поток
    [
    flag=1;
    for(a=1;flag<2;a=1) //оператор цикла
    {
        Sleep(500); //оператор ожидания создает паузу в 500 миллисекунд
        ff="!!!!!!!!!!!!!!!!!!";
    }
    ]
}

OnEvent("MACRO","2","RUN") //при запуске макрокоманды 2
{
    flag=2;
}
```

Пример использования событий и реакций объекта **Временная зона**:

1. При активировании первой временной зоны вывести на монитор видеоизображение с камеры 1.

```
OnEvent("TIME_ZONE","1","ACTIVATE")
{
    DoReact("CAM", "1", "ACTIVATE", "MONITOR<1>");
}
```

8.3.6 Примеры с Поворотными устройствами (PTZ) и Устройствами управления

TELEMETRY Поворотное устройство

TELEMETRY_EXT Пульт управления

JOYSTICK Устройство управления

8.3.6.1 Форматы

Формат процедуры событий для объекта **Поворотное устройство**:

```
OnEvent("TELEMETRY","_id_","_событие_")
```

Формат оператора для описания действий с **Поворотными устройствами**:

```
DoReact("TELEMETRY","_id_","_команда_" [,"_параметры_"]);
```

Формат процедуры событий для объекта **Пульт управления**:

```
OnEvent("TELEMETRY_EXT","_id_","_событие_")
```

Формат оператора для описания действий с **Пультом управления**:

```
DoReact("TELEMETRY_EXT","_id_","_команда_" [,"_параметры_"]);
```

Формат процедуры событий для объекта **Устройство управления**:

```
OnEvent("JOYSTICK","_id_","_событие_")
```

8.3.6.2 Примеры

Примеры использования реакций объекта **Поворотное устройство**:

1. Необходимо установить автофокусирование, когда видеокамеру 1 ставят на охрану.

```
OnEvent("CAM","1","ARM")
{
    DoReact("TELEMETRY","1", "AUTOFOCUS_ON");
}
```

2. Необходимо повернуть видеокамеру в положение, заданное в первом пресете, при включении реле.

```
OnEvent("GRELE","1","ON")
{
    telemetry_id= GetObjectParam("CAM","1","parent_id");
    DoReact("TELEMETRY","telemetry_id","SETUP","GO_preset<1>");
}
```

3. Записать маршрут патрулирования для Камеры 1, соответствующей Поворотному устройству 1.1. Маршрут состоит из двух точек, таких, что для перехода из точки 1 в точку 2 необходимо поворачивать камеру влево со скоростью 6 в течение 2 секунд. Патрулирование должно осуществляться со скоростью 10. Время нахождения в каждой точке маршрута – 25 секунд. Предполагается, что в момент начала выполнения программы камера установлена в положение, соответствующее первой точке маршрута.

```
OnEvent("MACRO","1","RUN")
{
    DoReact("TELEMETRY","1.1","PATROL_LEARN","cam<1>,preset<1>,tel_prior<1>,dwe
ll<25>,speed<10>,flush_tour<0>");
    Wait(2);
    DoReact("TELEMETRY","1.1","LEFT","speed<6>,tel_prior<1>");
    Wait(2);
    DoReact("TELEMETRY","1.1","STOP","speed<6>,tel_prior<1>");
    Wait(2);
    DoReact("TELEMETRY","1.1","PATROL_LEARN","cam<1>,preset<2>,tel_prior<1>,dwe
ll<25>,speed<10>,flush_tour<1>");
}
```

4. Есть 2 камеры с поворотными устройствами. Каждые 15 минут нужно повернуть камеры в пресет №1 (предустановка №1) и сделать скриншот. Имя файла – текущее время.

```
OnTime(W,D,X,Y,H,M, "01")
{
    if(strequal(M,"0"))
    {
        name=H+"_"+M+"_"+S+".jpg";
        //Камера 1 Поворотное устройство 1.1
        name1="Камера1 "+name;
        DoReact("TELEMETRY","1.1","GO_PRESET","preset<1>,tel_prior<1>");
        DoReact("MONITOR","1","EXPORT_FRAME","cam<1>,file<d:\"+name1);
        //Камера 2 Поворотное устройство 1.2
        name="Камера2 "+name;
        DoReact("TELEMETRY","1.2","GO_PRESET","preset<1>,tel_prior<1>");
        DoReact("MONITOR","1","EXPORT_FRAME","cam<2>,file<d:\"+name);
    }

    if(strequal(M,"15"))
    {
        name=H+"_"+M+"_"+S+".jpg";
        //Камера 1 Поворотное устройство 1.1
        name1="Камера1 "+name;
        DoReact("TELEMETRY","1.1","GO_PRESET","preset<1>,tel_prior<1>");
        DoReact("MONITOR","1","EXPORT_FRAME","cam<1>,file<d:\"+name1);
        //Камера 2 Поворотное устройство 1.2
        name="Камера2 "+name;
        DoReact("TELEMETRY","1.2","GO_PRESET","preset<1>,tel_prior<1>");
        DoReact("MONITOR","1","EXPORT_FRAME","cam<2>,file<d:\"+name);
```

```
    }

    if(strequal(M,"30"))
    {
        name=H+"_"+M+"_"+S+".jpg";
        //Камера 1 Поворотное устройство 1.1
        name1="Камера1 "+name;
        DoReact("TELEMETRY","1.1","GO_PRESET","preset<1>,tel_prior<1>");
        DoReact("MONITOR","1","EXPORT_FRAME","cam<1>,file<d:\"+name1);
        //Камера 2 Поворотное устройство 1.2
        name="Камера2 "+name;
        DoReact("TELEMETRY","1.2","GO_PRESET","preset<1>,tel_prior<1>");
        DoReact("MONITOR","1","EXPORT_FRAME","cam<2>,file<d:\"+name);
    }

    if(strequal(M,"45"))
    {
        name=H+"_"+M+"_"+S+".jpg";
        //Камера 1 Поворотное устройство 1.1
        name1="Камера1 "+name;
        DoReact("TELEMETRY","1.1","GO_PRESET","preset<1>,tel_prior<1>");
        DoReact("MONITOR","1","EXPORT_FRAME","cam<1>,file<d:\"+name1);
        //Камера 2 Поворотное устройство 1.2
        name="Камера2 "+name;
        DoReact("TELEMETRY","1.2","GO_PRESET","preset<1>,tel_prior<1>");
        DoReact("MONITOR","1","EXPORT_FRAME","cam<2>,file<d:\"+name);
    }
}
```

5. Осуществить патрулирование нескольких зон видимости с помощью пресетов поворотной камеры, с возможностью включения детектора движения на определенных областях этих зон.

   Камера 1: 5 зон детектора, 5 предустановок (пресетов). Два этих параметра задаются переменной n. Макрокоманда №1 – старт алгоритма. Макрокоманда №2 – остановка алгоритма. Flag – внутренняя переменная.

   При старте алгоритма камера становится в 1 пресет и ставит на охрану 1 зону детектора. Между этими командами задержка 200 миллисекунд, чтобы камера успела встать в пресет. Далее через 5 секунд 1 зона снимается с охраны и цикл начинается заново, но уже с 2 зоной и 2 пресетом. И так далее пока не переберутся все n зон и пресетов. После алгоритм начинается заново с 1. Алгоритм останавливается, если переменная flag обнуляется (с помощью макрокоманды №2).

```
OnEvent("MACRO","1","RUN")
{
    flag=1;
    n=5;
    [
     for(i=1;flag;i=str(i+1))
     {
```

```
            DoReact("TELEMETRY","1.1","GO_PRESET","preset<"+i+">,tel_prior<3>");
            Sleep(200);
            DoReact("CAM_ZONE","1"+i,"ARM");
            Wait(5);
            DoReact("CAM_ZONE","1"+i,"DISARM");
            if(strequal(i,n)) {i=0;}
        }
        ]
}

OnEvent("MACRO","2","RUN")
{
    flag=0;
}
```

Пример использования событий и реакций объекта **Пульт управления**:

По нажатию клавиши 15 на клавиатуре *AXIS T8312* включить на ней лампочку и поставить камеру 2 на охрану.

```
OnEvent ("TELEMETRY_EXT","1","KEY_PRESSED")
{
    if (strequal(param0, "15")){
        DoReact("TELEMETRY_EXT","1","RELE_ON","rele<15>");
        DoReact("CAM","2","ARM");
    }
}
```

8.3.7 Пример с Ядром

CORE Ядро

Запуск процедуры происходит при возникновении соответствующего события. Формат процедуры событий для объекта **Ядро**:

```
OnEvent("CORE","_id_","_событие_")
```

Пример. При появлении лица в кадре выводить на Монитор 2 видеоизображение с соответствующей камеры. При исчезновении лица убирать с Монитора 2 видеоизображение с соответствующей камеры.

```
OnEvent("CORE",N,"DO_REACT")
{
    if (strequal(action,"SET_MARKRECT"))
```

```
    {
        DoReact("MONITOR","2","ADD_SHOW","cam<"+param5_val+">");
    }
    if (strequal(action,"DEL_MARKRECT"))
    {
        [
        Wait(2);
        DoReact("MONITOR","2","REMOVE","cam<"+param0_val+">");
        ]
    }
}
```

8.3.8 Примеры с Сервером и Менеджером инцидентов

INC_MANAGER Менеджер инцидентов

INC_SERVER Сервер инцидентов

Формат процедуры событий для объекта **Менеджер инцидентов**:

```
OnEvent("INC_MANAGER","_id_","_событие_")
```

Формат процедуры событий для объекта **Сервер инцидентов**:

```
OnEvent("INC_SERVER","_id_","_событие_")
```

Формат оператора для описания действий с объектом **Сервер инцидентов**:

```
DoReact("INC_SERVER","_id_","_команда_" [,"_параметры_"]);
```

8.3.9 Примеры с Протоколом оператора и Протоколом событий

OPERATORPROTOCOL Протокол оператора

EVENT_VIEWER Протокол событий

8.3.9.1 Форматы

Формат процедуры событий для объекта **Протокол оператора**:

```
OnEvent("OPERATORPROTOCOL","_id_","_событие_")
```

Формат оператора для описания действий с **Протоколом оператора**:

```
DoReact("OPERATORPROTOCOL","_id_","_команда_" [,"_параметры_"]);
```

Формат процедуры событий для объекта **Протокол событий**:

```
OnEvent("EVENT_VIEWER","_id_","_событие_")
```

Формат оператора для описания действий с **Протоколом событий**:

```
DoReact("EVENT_VIEWER","_id_","_команда_" [,"_параметры_"]);
```

8.3.9.2 Примеры

Примеры использования событий и реакций объекта **Протокол оператора**:

1. По макрокоманде 2 удалять из окна Протокола оператора 1 первую тревогу по Камере 3.

```
OnEvent ("MACRO","2","RUN")
{
    DoReact("OPERATORPROTOCOL","1","DEL_ALARM","objtype<CAM>,objid<3>,options<f
irst>");
}
```

2. По макрокоманде 2 скрыть в окне Протокола оператора 1 кнопки Тревожная ситуация, Подозрительная ситуация, Ложное срабатывание для события Снята с охраны от Камеры 12.

```
OnEvent ("MACRO","2","RUN")
{
    DoReact("OPERATORPROTOCOL","1","HIDE_BUTTON","button<alarm,suspicious,false
>,hide<1>,objtype<CAM>,objaction<DISARM>,objid<12>");
}
```

Пример использования событий и реакций объекта **Протокол событий**:

По макрокоманде 1 для Протокола событий 1 устанавливается основной цвет фона черный, а основной цвет текста – белый.

```
OnEvent ("MACRO","1","RUN")
{
    DoReact("EVENT_VIEWER","1","UPDATE_VIEW","bk_color<#000000>, defclr<#FFFFFF>");
}
```

8.3.10 Примеры с Окном запроса оператора и SIP-терминалом

DIALOG Окно запроса оператора

SIP_TERMINAL SIP-терминал

8.3.10.1 Форматы

Формат оператора для описания действий с **Окном запроса оператора**:

```
DoReact("DIALOG","_id_","_команда_" [,"_параметры_"]);
```

Формат процедуры событий для объекта **SIP-терминал**:

```
OnEvent("SIP_TERMINAL","_id_","_событие_")
```

Формат оператора для описания действий с **SIP-терминалом**:

```
DoReact("SIP_TERMINAL","_id_","_команда_" [,"_параметры_"]);
```

8.3.10.2 Примеры

Примеры использования реакций объекта **Окно запроса оператора**:

1. Необходимо по макрокоманде с номером 1 устанавливать координаты верхнего левого угла окна запроса оператора (поворотной видеокамеры PANASONIC-850) в центре экрана, запрещать его перемещение и выводить его на экран.

```
OnEvent("MACRO","1","RUN")
{
    DoReact("DIALOG","PANASONIC-850","SETUP","x<50>,y<50>,allow_move<0>");
    DoReact("DIALOG","PANASONIC-850","RUN");
}
```

2. Необходимо закрывать окно запроса оператора по макрокоманде с номером 2.

```
OnEvent("MACRO","2","RUN")
{
    DoReact("DIALOG","PANASONIC-850","CLOSE");
}
```

8.3.11 Примеры с Аудио

PLAYER Аудиопроигрыватель

OLXA_LINE Микрофон

8.3.11.1 Форматы

Формат оператора для описания действий с **Аудиопроигрывателем**:

```
DoReact("PLAYER","_id_","_команда_" [,"_параметры_"]);
```

Формат процедуры событий для **Микрофона**:

```
OnEvent("OLXA_LINE", "_id_","_событие_")
```

Формат оператора для описания действий с **Микрофоном**:

```
DoReact("OLXA_LINE","_id_","_команда_" [,"_параметры_"]);
```

Функция проверки состояния объекта **Микрофон**:

```
CheckState("OLXA_LINE","номер","состояние")
```

8.3.11.2 Примеры

Примеры использования событий и реакций объекта **Аудиопроигрыватель**:

1. Необходимо проигрывать звуковой файл при остановке записи видеокамеры:

```
OnEvent("CAM",N,"REC_STOP")
{
    DoReact("PLAYER","1","PLAY_WAV","file<C:\Program Files
(x86)\Intellect\Wav\cam_alarm_"+N+".wav>,from_macro<1>");
}
```

2. Необходимо завершать проигрывание файла при начале записи видеокамеры:

```
OnEvent("CAM",N,"REC")
{
    DoReact("PLAYER","1","STOP_WAV");
}
```

3. Проигрывание звукового файла от прихода одного события до прихода другого события (в данном примере это запуск макрокоманд).

   Звуковой файл должен длиться не больше количества секунд, которое указано в операторе Wait.

```
OnEvent("MACRO","1","RUN")

{
    flag=1;
    [
    for(i=1;flag;i=1)
    {
        DoReact("PLAYER","1","PLAY_WAV","file<C:\Program
Files\Intellect\Wav\cam_alarm_1.wav>");
        Wait(3);
    }
    ]
}

OnEvent("MACRO","8","RUN")
{
    flag=0;
}
```

Примеры использования событий и реакций объекта **Микрофон**:

1. Необходимо включить первый микрофон на запись при включении акустопуска.

```
OnEvent("OLXA_LINE","1","accu_start") //включение акустопуска
{
    DoReact("OLXA_LINE","1","ARM"); //включение микрофона на запись
}
```

2. Необходимо установить минимальную компрессию на микрофоне при выключении записи аудиосигнала.

```
OnEvent("OLXA_LINE","1","DISARM") //отключение записи с микрофона
{
    DoReact("OLXA_LINE","1","SETUP","compression<5>"); //установлена минимальная компрессия
}
```

3. Микрофон (OLXA_LINE) пишется не синхронно с камерой. По умолчанию микрофон не стоит на охране. Необходимо писать звук как по акустопуску, так и по детекции от камеры. На сработку акустопуска (ACCU_START) и детектора движения (MD_START) включается принудительная запись звука и увеличивается на единицу переменная flag. При окончании акустопуска и детекции движения переменная flag уменьшается на единицу и запись звука останавливается, только если она равна нулю, т.е. нет ни акустопуска, ни движения.

```
OnInit()
{
    flag=0;
}

OnEvent("CAM","3","MD_START")
{
    flag=str(flag+1);
    DoReact("OLXA_LINE","1","RECORD_START");
}

OnEvent("OLXA_LINE","1","ACCU_START")
{
    flag=str(flag+1);
    DoReact("OLXA_LINE","1","RECORD_START");
}

OnEvent("OLXA_LINE","1","ACCU_STOP")
{
    flag=str(flag-1);
    if (!(flag))
    {
        DoReact("OLXA_LINE","1","RECORD_STOP");
    }
}

OnEvent("CAM","3","MD_STOP")
{
    flag=str(flag-1);
    if (!(flag))
    {
        DoReact("OLXA_LINE","1","RECORD_STOP");
    }
```

```
}
```

8.3.12 Пример с Видеошлюзом

GATE Видеошлюз

Формат процедуры событий для объекта **Видеошлюз**:

```
OnEvent("GATE","id","событие")
```

Формат оператора для действий с **Видеошлюзом**:

```
DoReact("GATE","id","команда"[,"параметры"]);
```

Пример. При падении темпа ввода на шлюзе 1 разослать соответствующее событие по всем ядрам системы.

```
OnEvent("GATE ","1"," GATE_LOW_FPS ")
{
    NotifyEventGlobal ("GATE ","1"," GATE_LOW_FPS ");
}
```

8.3.13 Пример с Детекторами

CAM_VMDA_DETECTOR Детектор VMDA

CAM_FACECAPTURE Детектор лиц

CAM_IP_DETECTOR Детектор встроенный

8.3.13.1 Форматы

Формат процедуры событий для **Детектора VMDA**:

```
OnEvent("CAM_VMDA_DETECTOR","_id_","_событие_")
```

Формат оператора для описания действий с **Детектором VMDA**:

```
DoReact("CAM_VMDA_DETECTOR","_id_","_команда_");
```

Формат событий для объекта **Детектор лиц**:

```
OnEvent("CAM_FACECAPTURE","_id_","_событие_")
```

Формат оператора для описания действий с **Детектором лиц**:

```
DoReact("CAM_FACECAPTURE","_id_","_команда_" ["_параметры_"]);
```

Формат процедуры событий для объекта **Детектор встроенный**:

```
OnEvent("CAM_IP_DETECTOR","_id_","_событие_")
```

8.3.13.2 Пример

Пример использования событий и реакций объекта **Детектор VMDA**:

При выполнении Макрокоманды 1 поставить на охрану Детектор VMDA 2:

```
OnEvent("MACRO","1","RUN")
{
DoReact("CAM_VMDA_DETECTOR","2","ARM");
}
```

8.3.14 Пример с Пользователем

Формат процедуры событий для объекта **Пользователь**:

```
OnEvent("PERSON","_id_","_событие_")
```

8.3.15 Примеры с Титрами

TITLEVIEWER Поиск по титрам

CAM_TITLE Титрователь

8.3.15.1 Форматы

Формат процедуры событий для объекта **Поиск по титрам**:

```
OnEvent("TITLEVIEWER","_id_","_событие_")
```

Формат оператора для описания действий с **Титрователем**:

```
DoReact("CAM_TITLE","_id_","_команда_");
```

8.3.15.2 Примеры

Пример для **Поиска по титрам**:

При двойном клике по строке результата поиска в окне Поиск по титрам отображать на мониторе 4 видеоархив, соответствующий данному результату.

```
OnEvent("TITLEVIEWER","1","GO_VIDEO")
{
    DoReact("MONITOR","4","ARCH_FRAME_TIME","cam<"+cam+">,date<"+date+">,time<"+time+"
>");
    DoReact("MONITOR","4","KEY_PRESSED","key<PLAY>");
}
```

Пример для **Титрователя**:

Запускать обновление базы данных титров по макрокоманде 1.

```
OnEvent("MACRO","1","RUN")
{
    DoReact("CAM_TITLE", "2", "REINDEX");
}
```

8.3.16 Примеры со Службой перезагрузки системы и Сервисом отказоустойчивости

SSS_WATCHDOG Служба перезагрузки системы

FAILOVER Сервис отказоустойчивости

8.3.16.1 Форматы

Формат процедуры событий для объекта **Служба перезагрузки системы**:

```
OnEvent("SSS_WATCHDOG","_id_","_событие_")
```

Формат оператора для описания действий со **Службой перезагрузки системы**:

```
DoReact("SSS_WATCHDOG","_id_","_команда_" [,"_параметры_"]);
```

Формат процедуры событий для объекта **Сервис отказоустойчивости**:

```
OnEvent("FAILOVER","_id_","_событие_")
```

Формат оператора для описания действий с **Сервисом отказоустойчивости**:

```
DoReact("FAILOVER","_id_","_команда_" [,"_параметры_"]);
```

8.3.16.2 Пример

Пример использования событий и реакций объекта **Служба перезагрузки системы**:

При перезагрузке модуля активировать третью камеру на монитор 5.

```
OnEvent("SSS_WATCHDOG","1"," RESTART_PROCESS")
{
    DoReact("MONITOR", "5", " ACTIVATE_CAM", "CAM<3>")
}
```

8.3.17 Пример с BacNet

BACNET BacNet

Формат процедуры событий для объекта **BacNet**:

```
OnEvent("BACNET","_id_","_событие_")
```

Формат оператора для описания действий с объектом **BacNet**:

```
DoReact("BACNET","_id_","_команда_" [,"_параметры_"]);
```

8.3.18 Примеры с Реле и Лучами

GRELE Реле

GRAY Луч

8.3.18.1 Форматы и функции

Формат процедуры событий для **Реле**:

```
OnEvent("GRELE", "_id_","_событие_")
```

Формат оператора для описания действий с **Реле**:

```
DoReact("GRELE","_id_","_команда_");
```

Функция проверки состояния объекта **Реле**:

```
CheckState("GRELE","номер", "состояние")
```

Формат процедуры событий для **Луча**:

```
OnEvent("GRAY","_id_","_событие_")
```

Формат оператора для описания действий с **Лучом**:

```
DoReact("GRAY","_id_","_команда_");
```

Функция проверки состояния объекта **Луч**:

```
CheckState ("GRAY","номер","состояние")
```

8.3.18.2 Примеры

Пример использования событий и реакции объекта **Реле**:

Необходимо при потере связи с реле 1 включить реле 2.

```
OnEvent("GRELE","1","SIGNAL_LOST")
{
    DoReact("GRELE", "2", "ON");
}
```

Примеры использования событий и реакций объекта **Луч**:

1. Необходимо перевести второй луч на второй вход, если потеряна связь с первым лучом.

```
OnEvent("GRAY","1"," SIGNAL_LOST") //потеряна связь с первым лучом
{
    DoReact("GRAY","2","SETUP","chan<2>"); //луч на втором входе
}
```

2. Необходимо разомкнуть второй луч и поставить на запись с откатом первую видеокамеру, в случае, когда первый луч замкнут.

```
OnEvent("GRAY","1"," ON") //первый луч замкнут
{
    DoReact("GRAY","2","SETUP","type<1>"); //разомкнуть второй луч
    DoReact("CAM","1","REC","rollback<1>"); //запись с откатом с первой видеокамеры
}
```

8.3.19 Примеры с Сервисами сообщений и оповещений

MMS Сервис почтовых сообщений
MAIL_MESSAGE Почтовое сообщение
VMS Сервис голосовых сообщений
VNS Сервис голосового оповещения
SMS Сервис коротких сообщений
TELEGRAM Telegram бот

8.3.19.1 Форматы

Формат процедуры событий для **Сервиса почтовых сообщений**:

```
OnEvent("MMS","_id_","_событие_")
```

Формат оператора для описания действий с **Сервисом почтовых сообщений**:

```
DoReact("MMS","_id_","_команда_" [,"_параметры_"]);
```

Формат процедуры событий для **Почтового сообщения:**

```
OnEvent("MAIL_MESSAGE","_id_","_событие_")
```

Формат оператора для описания действий с **Почтовым сообщением**:

```
DoReact("MAIL_MESSAGE","_id_","_команда_" [,"_параметры_"]);
```

Формат оператора для описания действий с **Сервисом голосовых сообщений**:

```
DoReact("VMS","_id_","_команда_" [,"_параметры_"]);
```

Формат оператора для описания действий с **Сервисом голосового оповещения**:

```
DoReact("VNS","_id_","_команда_" [,"_параметры_"]);
```

Формат процедуры событий для объекта **Сервис коротких сообщений**:

```
OnEvent("SMS","_id_","_событие_")
```

Формат оператора для описания действий с **Сервисом коротких сообщений**:

```
DoReact("SMS","_id_","_команда_" [,"_параметры_"]);
```

Формат процедуры событий для **Telegram бота**:

```
OnEvent("TELEGRAM","_id_","_событие_")
```

Формат оператора для описания действий с **Telegram ботом**:

```
DoReact("TELEGRAM","_id_","_команда_" [,"_параметры_"]);
```

8.3.19.2 Примеры

Пример использования реакций объекта **Сервис почтовых сообщений:**

Необходимо установить номер порта почтовой службы равным 25 при выполнении макрокоманды 1.

```
OnEvent("MACRO","1","RUN")
{
    DoReact("MMS", "1", "SETUP", "port<25>");
}
```

Пример использования реакций объекта **Почтовое сообщение**:

Необходимо отправить сообщение при срабатывании датчика движения вместе с изображением с видеокамеры при переходе видеокамеры в состояние тревоги.

```
OnInit(){
    i=0; //счетчик, используется для того чтобы избежать перезаписывания картинок с одной камеры
}

OnEvent("CAM",N,"REC") //видеокамера в состоянии тревоги

{
    filename = "c:\" + N + "_msg_"+str(i)+".jpg";
    i=i+1;
    DoReact("MONITOR","1","EXPORT_FRAME","cam<"+ N + ">,file<" + filename+ ">");
    DoReact("MAIL_MESSAGE", "1", "SETUP", "body<сработала камера"+ N + ">,
subject<тревога по камере>, from<server@itv.ru>, to<client@itv.ru>,attachments<" +
filename + ">");

    DoReact("MAIL_MESSAGE","1","SEND");
```

```
}
```

Пример использования реакций объекта **Сервис голосовых сообщений**:

Необходимо при выполнении макрокоманды 1 послать сообщение, если модем подключен к порту COM2, тип набора – импульсный, не дожидаться тонального сигнала.

```
OnEvent("MACRO","1","RUN")
{
    DoReact("VMS","1","SEND","modem<2>,pulse<1>,waitfordialtone<0>");
}
```

Примеры использования событий и реакций объекта **Сервис голосового оповещения**:

1. Необходимо проигрывать звуковой файл при остановке записи видеокамеры:

```
OnEvent("CAM",N,"REC_STOP")
{
    DoReact("VNS","1","PLAY","file<C:\Program Files
(x86)\Intellect\Wav\cam_alarm_"+N+".wav>");
}
```

2. Необходимо завершать проигрывание файла при начале записи видеокамеры:

```
OnEvent("CAM",N,"REC")
{
    DoReact("VNS","1","STOP");
}
```

3. Необходимо, чтобы при наступлении заранее заданной временной зоны менялось значение регулятора громкости на меньшее, а затем по её окончании, ставилось значение равному среднему.

```
OnEvent("TIME_ZONE","1","ACTIVATE")
{
    DoReact("VNS","1","SETUP","level<2>");
}
OnEvent("TIME_ZONE","1","DEACTIVATE")
{
    DoReact("VNS","1","SETUP","level<8>");
}
```

Примеры использования событий и реакций объекта **Сервис коротких сообщений**:

1. Необходимо послать короткое сообщение на номер «89179190909» при тревоге на первой видеокамере.

```
OnEvent("CAM","1","MD_START")
{
    DoReact("SMS","1","SETUP","phone<+79179190909>,message<камера 1, тревога>");
}
```

2. Необходимо установить устройство для передачи коротких сообщений и послать сообщение по номеру «89179190909» при тревоге на первом луче.

```
OnEvent("GRAY","1","CONFIRM") //принять тревогу от луча 1
{
    DoReact("SMS","1","SETUP","device<>",); //установить устройство для передачи
коротких сообщений
    DoReact("SMS","1","SETUP","phone<+79179190909>,message<луч  1, тревога>"); /
/послать сообщение о тревоге на луче 1 по номеру телефона
}
```

3. При получении SMS через Сервис почтовых сообщений 2 проиграть звуковой файл с: \Windows\Media\Tada.wav.

```
OnEvent("SMS","2","RECEIVE")
{
    DoReact("PLAYER","3","PLAY_WAV","file<c:\Windows\Media\Tada.wav>");
}
```

Примеры вызова команды для отправки сообщения в **Telegram** по макрокоманде:

```
OnEvent("MACRO","3","RUN") //запуск макрокоманды 3
{
  //Отправка с использованием chat_id и bot_id из настроек объекта:
  DoReact("TELEGRAM",1,"SEND","text<Hello world>");

  //Явное задание chat_id и bot_id при отправке:
  DoReact("TELEGRAM",1,"SEND","text<Hello
world>,chat_id<828752651>,bot_id<809045046:AAGtKxtDWu5teRGKW_Li8wFBQuJ-l4A9h38>");

  //Отправка файла с указанием идентификаторов чата и бота:
  DoReact("TELEGRAM",1,"SENDPHOTO","caption<Hello
world>,chat_id<828752651>,bot_id<809045046:AAGtKxtDWu5teRGKW_Li8wFBQuJ-
l4A9h38>,photo<G:\\1.jpg>");

  //Отправка геолокации с указанием идентификаторов чата и бота:
  DoReact("TELEGRAM",1,"SEND","text<Hello
world>,chat_id<828752651>,bot_id<809045046:AAGtKxtDWu5teRGKW_Li8wFBQuJ-
l4A9h38>",longtitude<37.3428359>,latitude<55.6841654>,address<ITV>);
}
```

8.4 Приложение 1. Приоритеты команд начала и остановки записи

Команды остановки и начала записи в ПК *Интеллект* могут иметь разный приоритет. Приоритет команд начала и остановки записи задается параметром priority<> реакций REC и REC_STOP соответственно. В случае, если производится попытка остановить запись командой с меньшим приоритетом, чем у команды, инициировавшей запись, команда на остановку записи будет проигнорирована.

При начале или остановке записи вручную, макрокомандой или по срабатыванию детектора приоритет не указывается явно. В таблице описано поведение ПК *Интеллект* при использовании разных способов начала и остановки записи.

Способ начала/остановки записи 1	**Способ начала/остановки записи 2**	**Поведение**
Начало/остановка записи инициированы оператором с помощью контекстного меню камеры (запись/остановить запись) или макрокомандой	Начало записи по реакции CAM|1|REC|, остановка по реакции CAM|1|REC_STOP|	Команды начала и остановки записи первого и второго способа равноценны*
Начало/остановка записи инициированы оператором с помощью контекстного меню камеры (запись/остановить запись) или макрокомандой	Начало записи по реакции CAM|1|REC|priority<0>, остановка по реакции CAM|1|REC_STOP|priority<0>	Остановка записи первым способом останавливает запись, начатую вторым способом
Начало/остановка записи инициированы оператором с помощью контекстного меню камеры (запись/остановить запись) или макрокомандой	Начало записи по реакции CAM|1|REC|priority<1> , остановка по реакции CAM|1|REC_STOP|priority<1>	Команда остановки записи первым способом останавливает запись, начатую вторым способом
Начало/остановка записи инициированы оператором с помощью контекстного меню камеры (запись/остановить запись) или макрокомандой	Начало записи по реакции CAM|1|REC|priority<2>, остановка по реакции CAM|1|REC_STOP|priority<2>	Команды начала и остановки записи первого и второго способа равноценны*
Начало/остановка записи инициированы оператором с помощью контекстного меню камеры (запись/остановить запись) или макрокомандой	Запись/остановка записи инициирована детектором (например, базовым детектором движения)	Команда остановки записи первым способом останавливает запись, начатую вторым способом

Начало записи по реакции CAM|1|REC|priority<0>, остановка по реакции CAM|1|REC_STOP|priority<0>	Запись/остановка записи инициирована детектором (например, базовым детектором движения)	Команда остановки записи вторым способом останавливает запись, начатую первым способом
Начало записи по реакции CAM|1|REC|priority<1>, остановка по реакции CAM|1|REC_STOP|priority<1>	Запись/остановка записи инициирована детектором (например, базовым детектором движения)	Возможны следующие варианты: 1. Если камера на охране и осуществляется запись по команде CAM|1|REC|priority<1>, при этом происходит начало тревоги по камере, то после окончания тревоги по камере запись продолжается. После команды CAM|1|REC_STOP|priority<1> запись оканчивается. 2. Если камера на охране, инициирована тревога по камере, а затем послана команда CAM|1|REC|priority<1>, после чего тревога по камере закончилась, то запись продолжается. После команды CAM|1|REC_STOP|priority<1> запись оканчивается. 3. Если камера на охране, инициирована тревога по камере, а затем послана команда CAM|1|REC_STOP|priority<1> то запись продолжается, а по окончании тревоги по камере запись оканчивается. 4. Если камера на охране и послана команда CAM|1|REC|priority<1>, начнется запись по камере. Если после этого будет инициирована тревога по камере и будет послана команда CAM|1|REC_STOP|priority<1>, то запись продолжится.
Начало записи по реакции CAM|1|REC|priority<2>, остановка по реакции CAM|1|REC_STOP|priority<2>	Запись/остановка записи инициирована детектором (например, базовым детектором движения)	Команда остановки записи первого способа останавливает запись, начатую вторым способом

*Равноценность способов означает, что остановка записи способом 1 возможна, если запись инициирована способом 2, и наоборот, остановка записи способом 2 возможна, если запись инициирована способом 1

8.5 Приложение 2. Определение значений param_id и param_value для реакции SET_IPINT_PARAM

Значения параметров **param_id** и **param_value**, необходимых для использования реакции SET_IPINT_PARAM, могут быть индивидуальны как для каждой из интегрированных IP-камер, так и для их прошивок.

Определение значений **param_id** и **param_value** осуществляется следующим образом:

1. Открыть директорию с установленным DriverPack, по умолчанию **C:\Program Files\Common Files\ITV\Ipint.DriverPack\3.0.0**
2. Открыть с помощью любого текстового редактора содержащийся в данном каталоге файл с именем **Ipint.<Название драйвера камеры>.rep**, например Ipint.SonyIpela.rep

Примечание.

В большинстве случаев имя драйвера совпадает с названием производителя IP-устройства. Уточнить имя драйвера для требуемого производителя можно при обращении в техническую поддержку компании ITV.

3. Найти в файле название требуемой модели, например SNC-DH120T.

```
    <model>
        <brand>Sony</brand>
        <name>SNC-DH120T</name>
        <firmware>1.12.03</firmware>
        <firmware>1.74.01</firmware>
        <firmware>1.75.00</firmware>
    </model>
    <credentialsRef id="creds"/>
    <videoSourceRef id="video_source_dh160">
        <videoStreamingRef id="vs-5generation-megapixel-tvStandard" default="true"/>
        <videoStreamingRef id="vs-5generation-secondary-ch120"/>
        <detectorRef id="sony-detector-area-1280x1024" maxCount="1"/>
        <detectorRef id="sony-detector-tamper" maxCount="1"/>
    </videoSourceRef>
    <telemetryRef id="telemetry_5g"/>
    <ioDeviceRef id="iodev-sony-1ray-1relay"/>
</device>
```

4. В пределах того же тэга <device>, что и тэг <model>, содержащий описание требуемой модели, присутствует тэг **<videoSourceRef>**. Необходимо найти в файле еще одно вхождение значения **id** данного параметра (в данном примере это значение video_source_dh160) в тэге **videoSource**.

```
<videoSource id="video source dh160">
    <property id="brightness" xsi:type="PropertyIntRangeType">
        <value>
            <min>0</min>
            <max>10</max>
            <default>5</default>
        </value>
    </property>
    <property id="sharpness" xsi:type="PropertyIntRangeType">
        <value>
            <min>0</min>
            <max>6</max>
            <default>3</default>
        </value>
    </property>
    <property id="saturation" xsi:type="PropertyIntRangeType">
        <value>
            <min>0</min>
            <max>6</max>
            <default>3</default>
        </value>
    </property>
    <property id="contrast" xsi:type="PropertyIntRangeType">
        <value>
            <min>0</min>
            <max>6</max>
            <default>3</default>
        </value>
    </property>
    <property id="monochrome" xsi:type="PropertyBoolType" default="false"/>
    <property id="daynight" xsi:type="PropertyStringEnumType">
        <value default="true">auto</value>
        <value name="night">on</value>
        <value name="day">off</value>
        <value name="timer">timer</value>
        <value name="sensor">sensor</value>
    </property>
    <property id="dayNightAutoThreshold" xsi:type="PropertyStringEnumType">
        <value name="high" default="true">high</value>
        <value name="low">low</value>
    </property>
```

5. В тэгах **<property>** описаны параметры IP-устройства и их возможные значения. Способ описания возможных значений зависит от их типа.

В приведенном примере можно использовать, например, параметр **param_id**="daynight" для переключения режима камеры **День/Ночь**. В таком случае возможные значения параметра **param_value**: auto, on, off, timer или sensor.

Пример

Пример использования реакции SET_IPINT_PARAM:

1. Для объекта **Камера**:
   DoReact("CAM", "1","SET_IPINT_PARAM","param_id<daynight>,param_value<on>");
2. Для объекта **Устройство видеоввода**:
   DoReact("GRABBER",
   "1","SET_IPINT_PARAM","param_id<daynight>,param_value<on>,cam_id<1>");

Результатом выполнения обеих реакций будет установка значения параметра "daynight" для Камеры 1 равным "on".

Для работы реакции SET_IPINT_PARAM необходимо, чтобы в ПК *Интеллект* был активирован многопоточный режим (см. Руководство администратора, раздел Настройка многопоточного видеосигнала). При этом следует учитывать, что если для камеры интегрирован только один видеопоток, в многопоточном режиме не будет отображаться видеоизображение.

Узнать количество интегрированных потоков для камеры можно в списке IP-оборудования, интегрированного в ПК *Интеллект*, который находится на странице Documentation Drivers Pack.

Если данный способ неприменим по каким-либо причинам, количество интегрированных видеопотоков можно узнать следующим образом:

1. Повторить шаги 1-3 предыдущего алгоритма.
2. В пределах того же тэга <device>, в котором описана требуемая модель, в тэгах **<videoStreamingRef>** описаны интегрированные видеопотоки. Их должно быть больше одного.

```
    <model>
        <brand>Sony</brand>
        <name>SNC-DH120T</name>
        <firmware>1.12.03</firmware>
        <firmware>1.74.01</firmware>
        <firmware>1.75.00</firmware>
    </model>
    <credentialsRef id="creds"/>
    <videoSourceRef id="video_source_dh160">
        <videoStreamingRef id="vs-5generation-megapixel-tvStandard" default="true"/>
        <videoStreamingRef id="vs-5generation-secondary-ch120"/>
        <detectorRef id="sony-detector-area-1280x1024" maxCount="1"/>
        <detectorRef id="sony-detector-tamper" maxCount="1"/>
    </videoSourceRef>
    <telemetryRef id="telemetry_5g"/>
    <ioDeviceRef id="iodev-sony-1ray-1relay"/>
</device>
```

9 Объект Скрипт. Программирование с использованием языка JScript

9.1 Назначение и возможности языка JScript

9.2 Инструментарий программирования на JScript

- Системный объект Скрипт
- Утилита Редактор-Отладчик
- Отладочное окно
    - Включение Отладочного окна
    - Работа с Отладочным окном
        - Копирование информации о событии или реакции в буфер обмена
        - Выделение сообщений цветом
        - Фильтр событий и реакций
        - Поиск событий и реакций
        - Очистка Отладочного окна
- Получение списка системных названий объектов, реакций и событий ПК Интеллект

9.3 Создание первого скрипта

9.4 Работа со скриптом

- Создание скрипта
- Сохранение скрипта
- Удаление скрипта
- Поиск в скрипте
- Замена в скрипте

9.5 Отладка скриптов

- Возможности отладки скриптов
- Создание и использование тестовых событий

- Работа с отладочными окнами утилиты Редактор-Отладчик
  - Просмотр сообщений скрипта
  - Отображение сообщений о запуске, проверке, изменении и выполнении скриптов в отладочных окнах
- Использование сторонних программ-отладчиков

9.6 Примеры скриптов на языке JScript

- Примеры скриптов с Монитором видеонаблюдения и Камерами
- Примеры скриптов с Картой
- Примеры скриптов с Детекторами
- Примеры скриптов с Макрокомандами
- Пример скрипта с Пользователями
- Примеры скриптов с Сервером и Менеджером инцидентов
- Пример скрипта с Сервисом отказоустойчивости
- Примеры скриптов с BacNet
- Пример с ботом Telegram
- Примеры скриптов с Протоколом событий

9.7 Приложение 1. Описание утилиты Редактор-Отладчик

- Назначение утилиты Редактор-Отладчик
- Описание интерфейса утилиты Редактор-Отладчик
  - Интерфейс утилиты Редактор-Отладчик
  - Вкладка Скрипт отладка-редактирование
  - Вкладка Сообщения скрипта
  - Главное меню
    - Описание интерфейса главного меню
    - Описание пункта главного меню Файл
    - Описание пункта главного меню Вид
    - Описание пункта главного меню Отладка и редактирование
    - Описание элементов пункта главного меню Список сообщений
  - Описание диалогового окна Фильтр
  - Описание диалогового окна Выделить цветом
  - Описание панели инструментов утилиты Редактор-Отладчик

9.8 Приложение 2. Создание виртуальных объектов с возможностью задавать события, реакции и состояния

9.9 Назначение и возможности языка JScript

В составе программного комплекса *Интеллект* язык программирования JScript позволяет реализовывать дополнительные пользовательские функции, не предусмотренные основными функциональными возможностями программы.

Язык программирования JScript является стандартным средством разработки пользовательских скриптов. В программном комплексе *Интеллект* поддерживается версия языка JScript, реализованная в технологии ActiveX корпорации Microsoft. Описание объектной модели языка JScript, используемого в программном комплексе *Интеллект,* приведено в документации корпорации Microsoft (например, MSDN).

Интерпретация скриптов, разработанных на языке JScript для программного комплекса *Интеллект,* выполняется с использованием стандартных (входящих в поставку ОС Windows) программных компонентов ActiveX. Поэтому при разработке скриптов допускается использование любых компонентов объектной модели версии языка JScript, реализованной в технологии ActiveX.

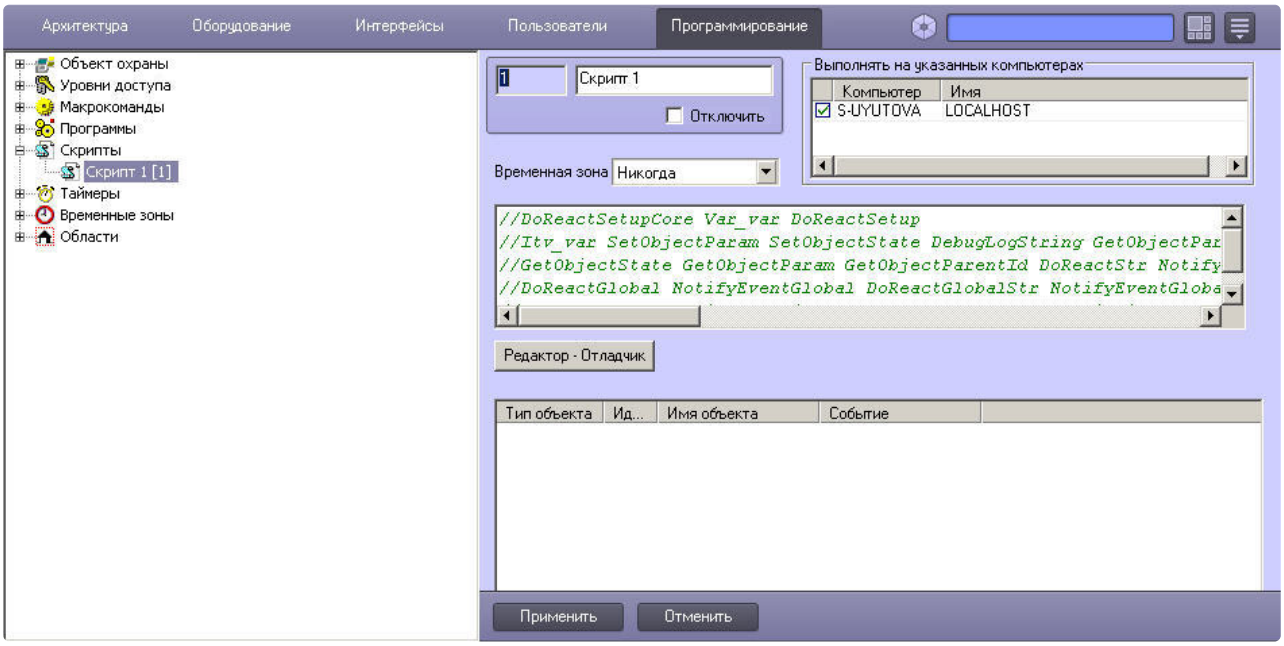
Программный комплекс *Интеллект* дополнительно предоставляет специализированную объектную модель для разработки скриптов на языке JScript, позволяющую работать с системными объектами программного комплекса *Интеллект*, получать и отправлять системные события и реакции.

9.10 Инструментарий программирования на JScript

9.10.1 Системный объект Скрипт

Системный объект **Скрипт** предназначен для инициализации в ПК *Интеллект* скрипта, разработанного на языке JScript, и задания параметров его выполнения.

Панель настройки системного объекта **Скрипт** представлена на рисунке:



Внимание!

Создание большого количества (более 100) объектов **Скрипт** может привести к нестабильной работе системы.

В панели настройки системного объекта **Скрипт** указываются временная зона выполнения скрипта и компьютеры (ядра), на которых требуется выполнять скрипт.

Примечание.

Для того, чтобы установить флажки напротив всех компьютеров, необходимо выделить ячейку в столбце с флажками и нажать Ctrl+A. Для снятия всех флажков необходимо выделить ячейку и нажать Shift+A.

На панели настройки системного объекта **Скрипт** размещены кнопка запуска утилиты *Редактор-Отладчик* и панель просмотра текста скрипта, созданного посредством данной утилиты. Редактирование скрипта может осуществляться с использованием утилиты *Редактор-Отладчик* или непосредственно из панели настройки объекта **Скрипт**.

Кроме того, имеется возможность настроить фильтр событий – список событий, которые будет обрабатывать системный объект **Скрипт**. Внесение события в фильтр равносильно оператору if в тексте скрипта, то есть при указании события в таблице данный оператор можно опустить.

Внимание!

Задание фильтра событий обязательно выполнять при создании скрипта в больших распределенных конфигурациях. В противном случае модуль будет обрабатывать все входящие события и может работать некорректно.

Пример.

Пусть в таблице в столбце **Тип объекта** указано значение **Макрокоманда**, в столбце **Идентификатор** выбрано значение **1**, в столбце **Событие** - **Выполнено действие**. Тогда вместо скрипта

```
if (Event.SourceType == "MACRO" && Event.SourceId==1 && Event.Action ==
"RUN")
{
    DoReactStr("CAM", "2", "REC", "");
}
```

можно использовать скрипт

```
DoReactStr("CAM", "2", "REC", "");
```

Подробно элементы панели настроек объекта **Скрипт** описаны в документе Руководство администратора.

Пример.

При управлении поворотной камерой из Монитора видеонаблюдения начинать запись по Камере 1.

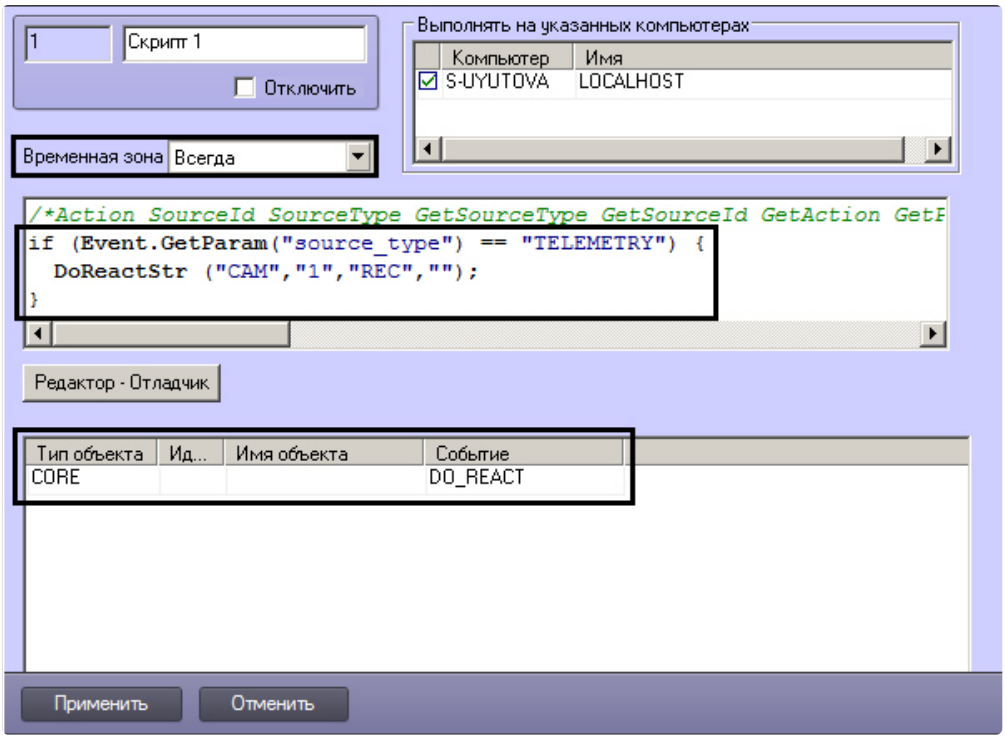
Для этого следует настроить объект **Скрипт** следующим образом:

1. Выбрать требуемую временную зону, когда должен выполняться скрипт.
2. Ввести текст скрипта:

```
if (Event.GetParam("source_type") == "TELEMETRY") {
  DoReactStr ("CAM","1","REC","");
}
```

3. Настроить фильтр следующим образом:
   a. Из раскрывающегося списка **Тип объекта** выбрать CORE.

b. В поле **Событие** ввести DO_REACT.



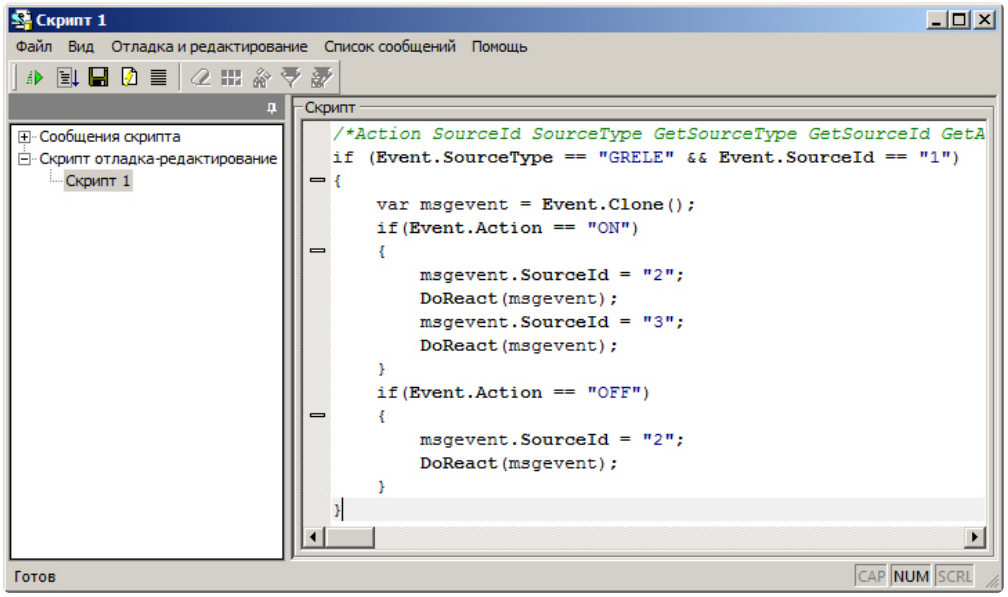
Заполнять данный фильтр можно также при помощи события UPDATE_OBJECT объекта CORE. Пример команды для добавления в фильтр объекта **Скрипт** с идентификатором 2 объекта **Камера** с идентификатором 1:

```
NotifyEventStr("CORE","","UPDATE_OBJECT","objtype<SCRIPT>,objid<2>,EVENT.objid.0<1>,E
VENT.objid.1<10>,EVENT.action.count<2>,flags<>,EVENT.action.0<>,EVENT.action.1<>,EVEN
T.objtype.0<CAM>,EVENT.objtype.count<2>,EVENT.objtype.1<CAM>,EVENT.objid.count<2>");
```

9.10.2 Утилита Редактор-Отладчик

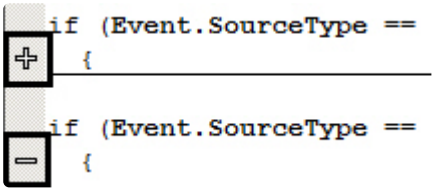
Утилита *Редактор-Отладчик* предназначена для создания, отладки и редактирования скриптов в программном комплексе *Интеллект*.

Диалоговое окно утилиты *Редактор-Отладчик* представлено на рисунке.



Утилита *Редактор-Отладчик* содержит встроенные текстовый редактор и отладочное окно.

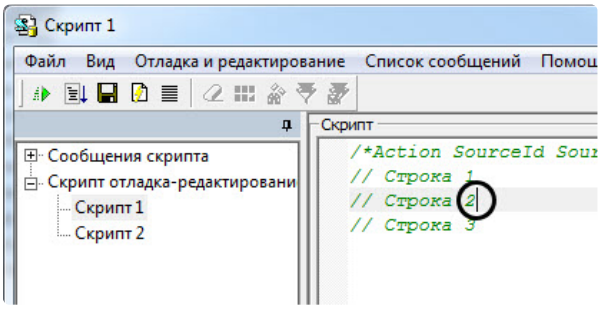
Для удобства контроля корректности написания скриптов в текстовом редакторе реализовано автоматическое выделение объектов, методов и свойств, входящих во встроенную в программный комплекс *Интеллект* объектную модель языка JScript. Также предусмотрена возможность сворачивания блоков кода с помощью кнопок **-** и **+** в левой части текстового поля **Скрипт**.



Примечание.

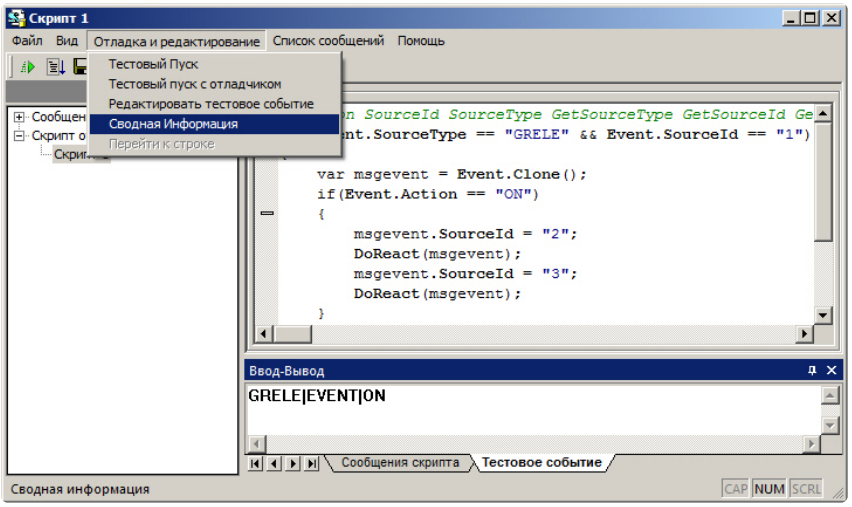
При переходе между скриптами, сообщениями скрипта и другими объектами в дереве утилиты *Редактор-Отладчик* настройки сворачивания блоков сбрасываются, т.е. все блоки в скрипте становятся развернутыми.

Позиция курсора в каждом скрипте запоминается в рамках одной сессии работы ПК *Интеллект* (при перезагрузке происходит сбрасывание позиции курсора). Например, если в **Скрипт 1** установить курсор в конец строки **//Строка 2**, затем переключиться на **Скрипт 2** и произвести в нем какие-либо действия, вернувшись обратно в **Скрипт 1**, курсор будет стоять также в конце строки **//Строка 2**.



Отладочное окно утилиты *Редактор-Отладчик* позволяет просматривать сведения обо всех событиях, регистрируемых в системе. Имеется возможность настроить фильтр событий, отображающихся в отладочном окне. Для каждого системного объекта **Скрипт** в утилите *Редактор-Отладчик* создается отдельное отладочное окно, что при использовании фильтров дает возможность отлаживать каждый скрипт независимо от других.

Для отладки скрипта предусмотрена возможность тестового запуска с использованием заданного пользователем тестового события, генерируемого утилитой и не регистрируемого в системе. Чтобы отобразить или редактировать это событие, следует выбрать **Отладка и редактирование > Сводная информация**, а затем перейти на вкладку **Тестовое событие** на открывшейся панели в нижней части окна. См. также Создание и использование тестовых событий.



Созданный скрипт может быть сохранен в системном объекте **Скрипт** или в текстовом файле на жестком диске компьютера.

9.10.3 Отладочное окно

В программном комплексе *Интеллект* существует возможность в реальном времени просматривать все события и реакции, происходящие в системе. События и реакции со свойствами объектов отображаются в **Отладочном окне**, откуда их можно скопировать в буфер обмена Windows для последующего использования в программах.

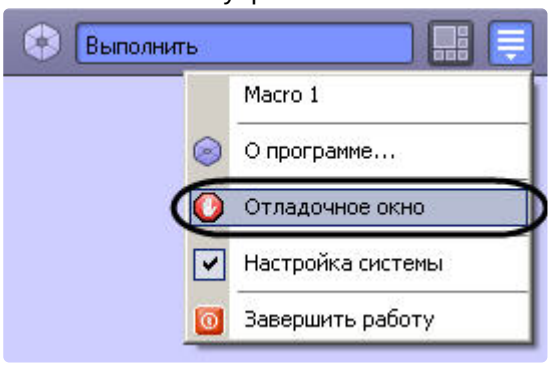
9.10.3.1 Включение Отладочного окна

Начиная с хотфикса 4.11.3.4472 по умолчанию включен режим **Debug 4**. На предыдущих версиях **Отладочное окно** по умолчанию выключено. Для включения **Отладочного окна** необходимо:

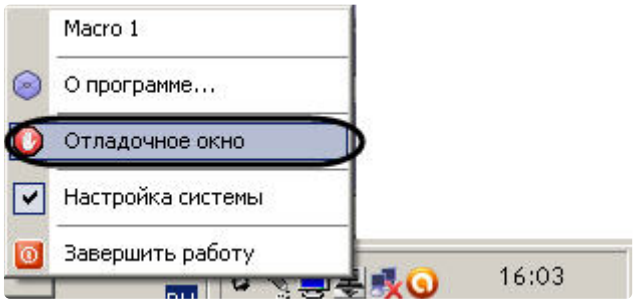
1. Завершить работу программного комплекса *Интеллект*.
2. Запустить утилиту **Расширенная настройка** *Tweaki.exe*.

> **Примечание.**
>
> Отладочное окно можно включить и без использования утилиты tweaki.exe. Для этого следует установить строковый параметр Debug равным 1, 2, 3 либо 4 в разделе HKEY_LOCAL_MACHINE\SOFTWARE\ITV\Intellect реестра ОС Windows (HKEY_LOCAL_MACHINE\Software\Wow6432Node\ITV\Intellect для 64-битной системы).

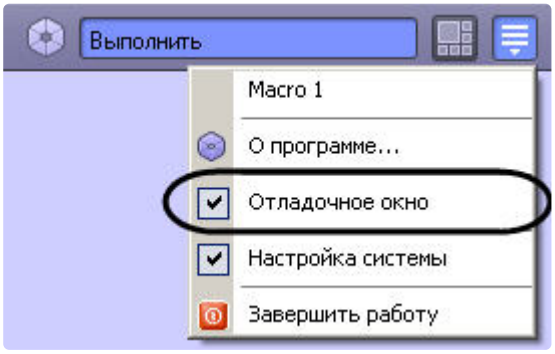
3. Выбрать раздел **Интеллект** в дереве, расположенном в левой части диалогового окна утилиты.
4. Изменить значение параметра **Режим отладки** с **Выключен** на **Debug 1**, **Debug 2**, **Debug 3** или **Debug 4**. Любой из этих режимов включит **Отладочное окно**, разница между ними в объёме записываемой информации в лог-файлы (см. Панель настройки раздела Интеллект).
5. Нажать кнопку **ОК**.
6. Запустить программный комплекс *Интеллект*.
7. В Главной панели управления ПК *Интеллект* появится новый пункт **Отладочное окно**.



> **Примечание.**
>
> Также данное меню доступно из области уведомлений (системного трея) Windows при нажатии левой кнопкой мыши на значок  или по кратковременному удержанию горячей клавиши F8.



8. Выбрать пункт **Отладочное окно** в Главной панели управления для отображения Отладочного окна на экране монитора. Выбранный пункт меню **Отладочное окно** будет отмечен флажком.



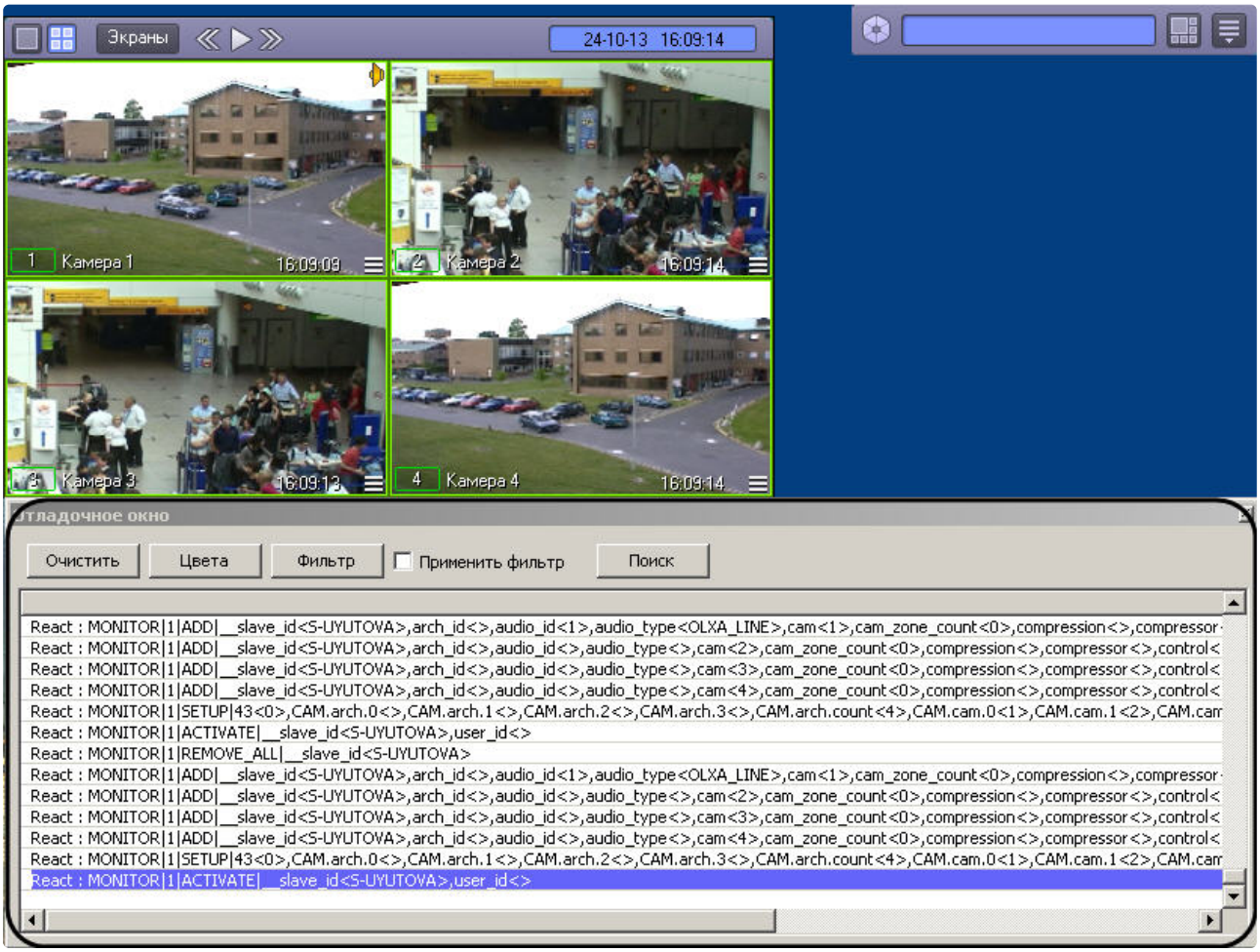
Для того, чтобы скрыть Отладочное окно, нужно повторно выбрать пункт **Отладочное окно** в Главной панели управления.

Примечание.

Для выключения Отладочного окна следует выбрать значение **Выключен** для параметра **Режим отладки** в утилите tweaki.exe, либо установить строковый параметр Debug равным 0 в разделе HKEY_LOCAL_MACHINE\SOFTWARE\ITV\Intellect реестра ОС Windows (HKEY_LOCAL_MACHINE\Software\Wow6432Node\ITV\Intellect для 64-битной системы). Данные операции проводятся при выгруженном ПК *Интеллект*.

9.10.3.2 Работа с Отладочным окном

Внешний вид Отладочного окна представлен на рисунке. В Отладочном окне выводится последовательность событий и реакций в системе.



Отладочное окно обладает следующими свойствами:

1. располагается поверх других окон;
2. для изменения размеров Отладочного окна следует использовать мышь;
3. существует возможность копировать информацию о событии или реакции в буфер обмена Windows для последующего использования в программах;
4. существует возможность фильтровать события или реакции, отображаемые в отладочном окне;
5. существует возможность выделять цветом события или реакции, отображаемые в Отладочном окне;
6. существует возможность поиска событий или реакций в Отладочном окне.

При выделении цветом и фильтрации сообщений отладочного окна могут быть использованы регулярные выражения.

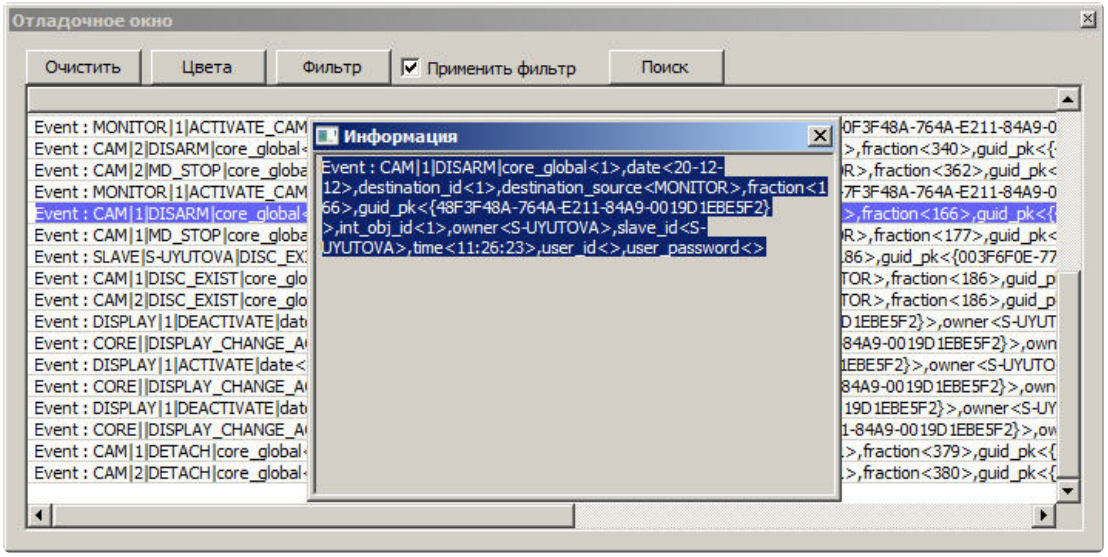
9.10.3.2.1 Копирование информации о событии или реакции в буфер обмена

Чтобы прочитать и/или скопировать в буфер обмена Windows информацию о событии или реакции, необходимо выполнить следующую последовательность действий:

1. Выделить требуемую строку в **Отладочном окне**.
2. Кликнуть правой кнопкой мыши по выделенной строке. В результате появится окно **Информация**, содержащее информацию о требуемом событии или реакции.
3. Для копирования в буфер обмена Windows выделить требуемые сведения, после чего нажать **Ctrl+C**.

> **Примечание.**
>
> Для операций с текстом в окне **Информация** удобно использовать контекстное меню (вызывается кликом правой кнопкой мыши по выделенному тексту).

4. Для закрытия окна **Информация** необходимо нажать .

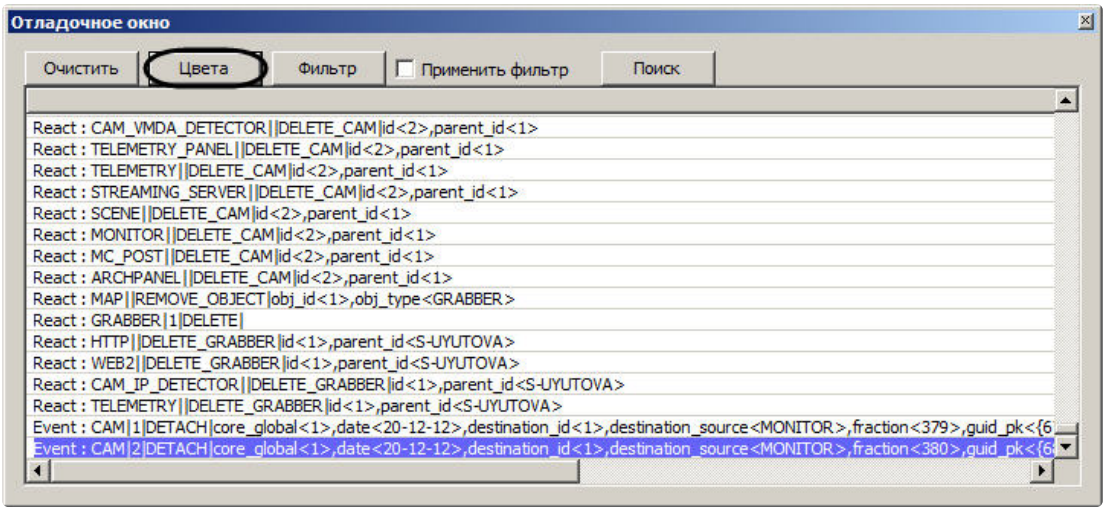


Копирование информации о событии или реакции в буфер обмена Windows завершено.

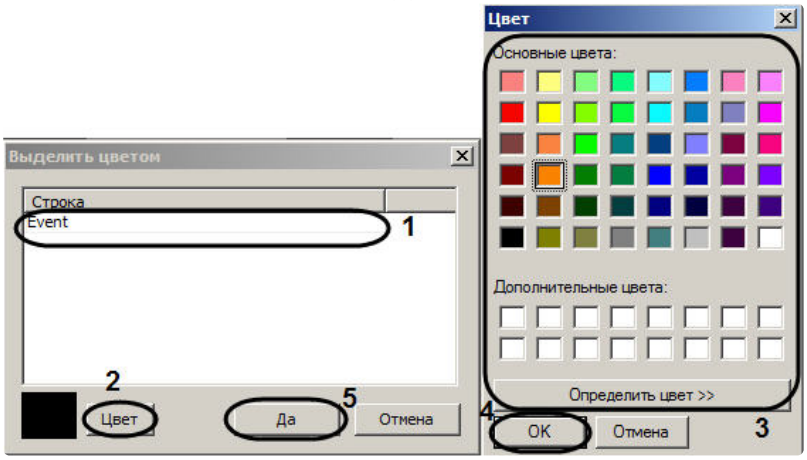
9.10.3.2.2 Выделение сообщений цветом

Для того чтобы настроить выделение сообщений в отладочном окне цветом, необходимо:

1. Нажать на кнопку **Цвета**.



2. В открывшемся окне **Выделить цветом** ввести строку, при наличии которой в сообщении оно должно быть выделено цветом (1).

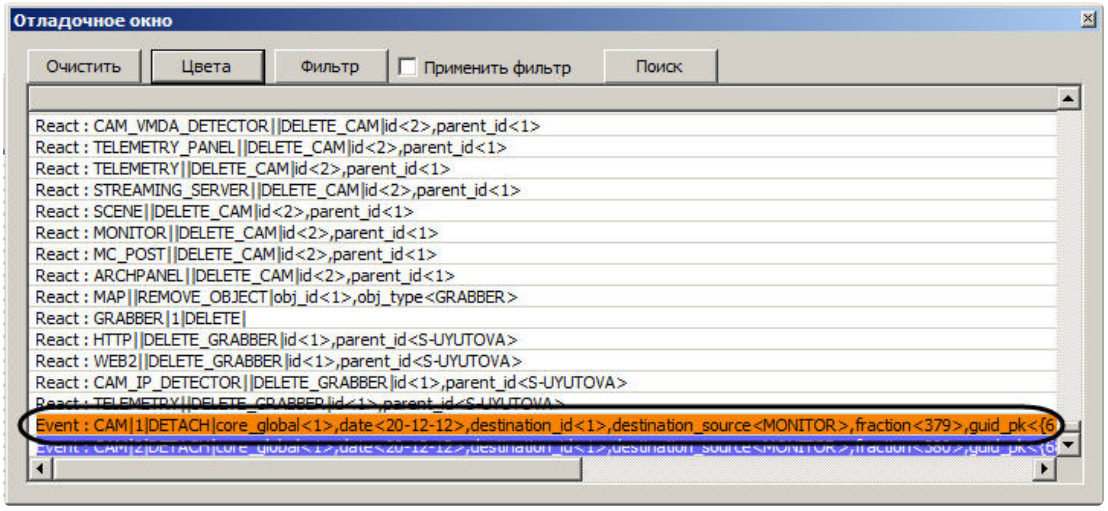


3. Нажать на кнопку **Цвет** (2).
4. В стандартном диалоговом окне Windows **Цвет** выбрать цвет подсветки сообщения (3).
5. Нажать на кнопку **ОК** (4).
6. Повторить шаги 2-5 для всех требуемых строк.

> **Примечание.**
>
> Для добавления строки в таблицу следует нажать на клавишу "вниз" на клавиатуре.

7. Нажать на кнопку **Да** (5).

В результате в **Отладочном окне** будут подсвечены выбранным цветом сообщения, включающие введенную строку.



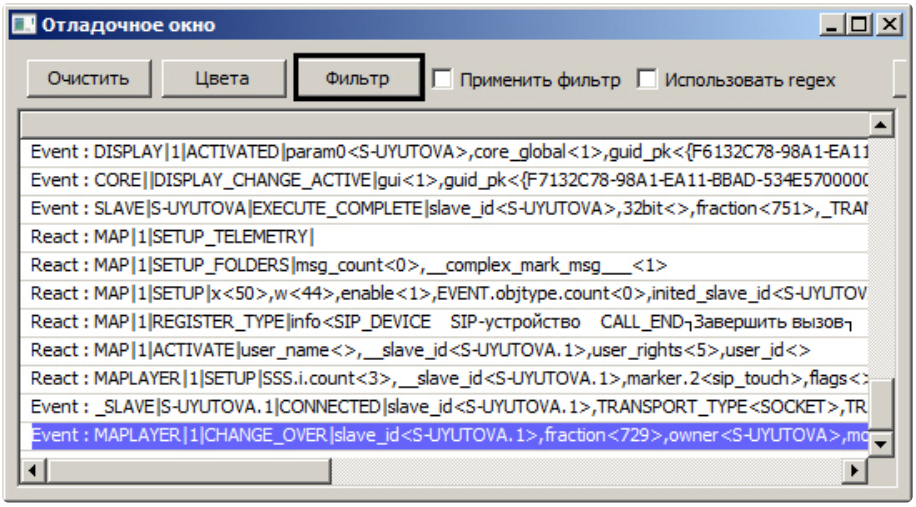
Выделение сообщений цветом завершено.

9.10.3.2.3 Фильтр событий и реакций

Фильтр событий и реакций позволяет отображать в **Отладочном окне** только требуемые сообщения.

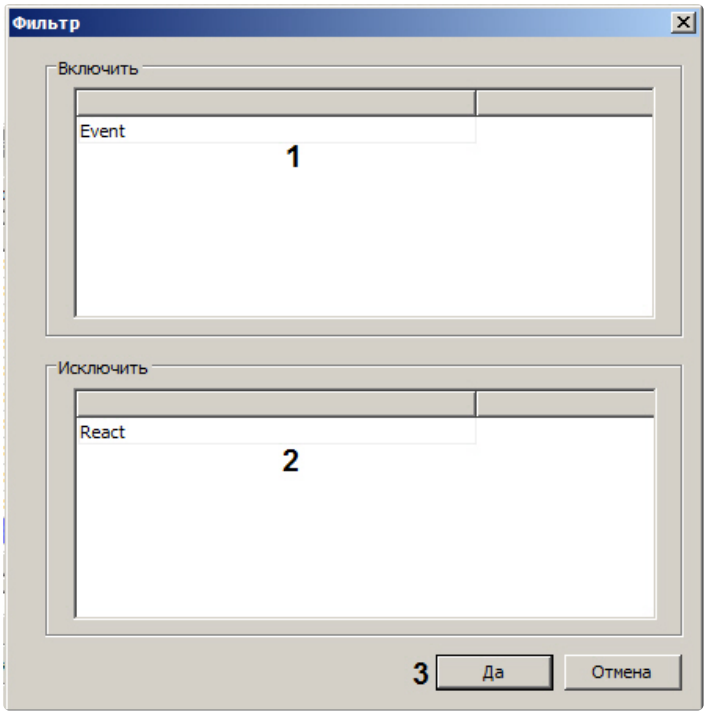
Для настройки фильтра событий и реакций необходимо выполнить следующие действия:

1. Нажать на кнопку **Фильтр.**



2. В открывшемся окне **Фильтр** указать строки, которые должны содержаться в сообщении, чтобы оно было отображено в **Отладочном окне** (1). Можно использовать регулярные выражения, в

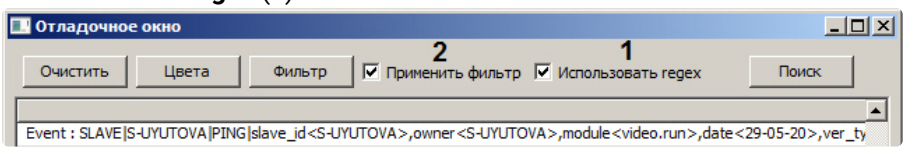
этом случае следует установить флажок **Использовать regex** на шаге 5.



3. Указать строки, при наличии которых в сообщении оно не отображается в **Отладочном окне** (2). Можно использовать регулярные выражения, в этом случае не следует забывать установить флажок **Использовать regex** на шаге 5.

> **Примечание.**
>
> Для добавления строки в таблицу следует нажать на кнопку "вниз" на клавиатуре.

4. Нажать на кнопку **Да** (3).
5. Если в строках фильтра применялись регулярные выражения, установить флажок **Использовать regex** (1).



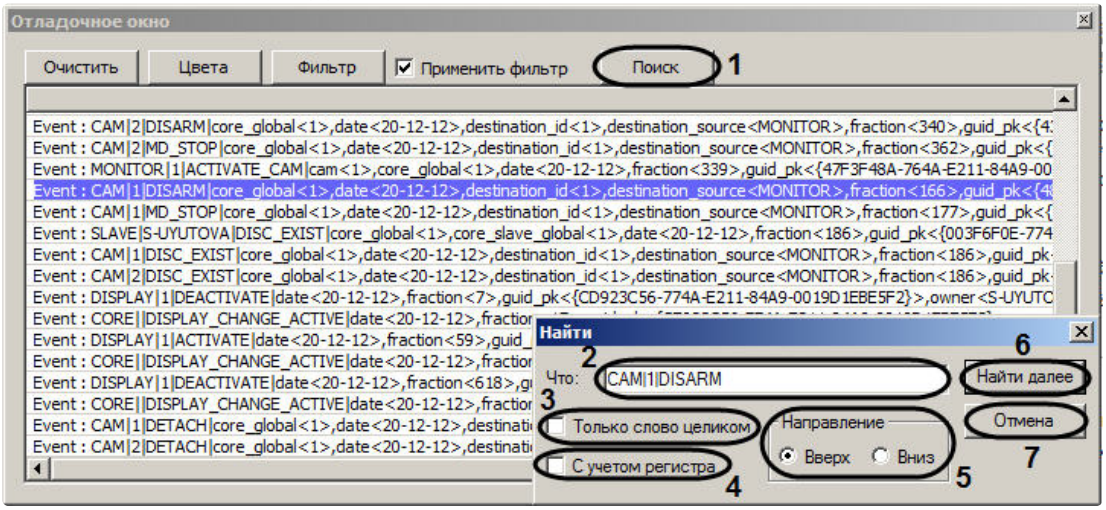
6. Для применения фильтра установить флажок **Применить фильтр** (2).
   В результате в отладочном окне будут отображены только сообщения, удовлетворяющие условиям фильтра.

Настройка фильтра событий и реакций завершена.

9.10.3.2.4 Поиск событий и реакций

Поиск событий и реакций осуществляется следующим образом:

1. Нажать на кнопку **Поиск** (1). Будет открыто окно **Найти**.



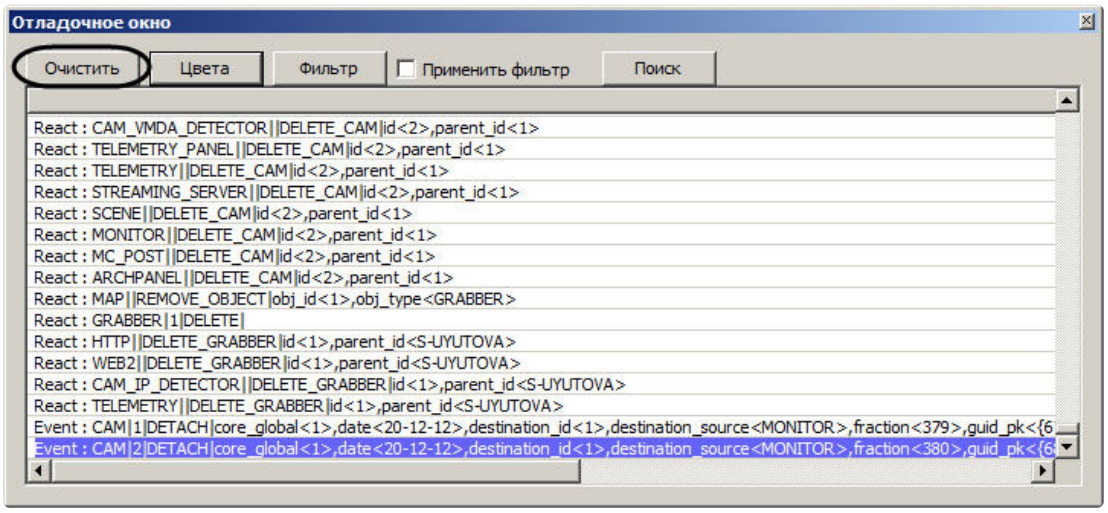
2. Ввести в поле **Что** условие поиска (2).
3. Если требуется искать введенную строку как самостоятельное слово, не присутствующее в других словах в виде части, а отделенное от них как минимум одним пробелом, необходимо установить флажок **Только слово целиком** (3).
4. Если при поиске следует учитывать регистр символов, необходимо установить флажок **С учетом регистра** (4).
5. Установить переключатель **Направление** в положение, соответствующее направлению поиска (5).
6. Для просмотра следующего результата поиска нажать на кнопку **Найти далее** (6).

> **Примечание.**
>
> Для закрытия окна **Найти** нажать на кнопку **Отмена.**

Поиск событий и реакций завершен.

9.10.3.2.5 Очистка Отладочного окна

Для того чтобы удалить из **Отладочного окна** все сообщения необходимо нажать на кнопку **Очистить**.



9.10.4 Получение списка системных названий объектов, реакций и событий ПК Интеллект

Список системных названий объектов, реакций и событий программного комплекса *Интеллект*, используемых при программировании, можно получить при помощи утилиты настройки конфигурации *ddi.exe*. Реакции основных объектов системы описаны в разделе Описание событий и реакций объектов системы.

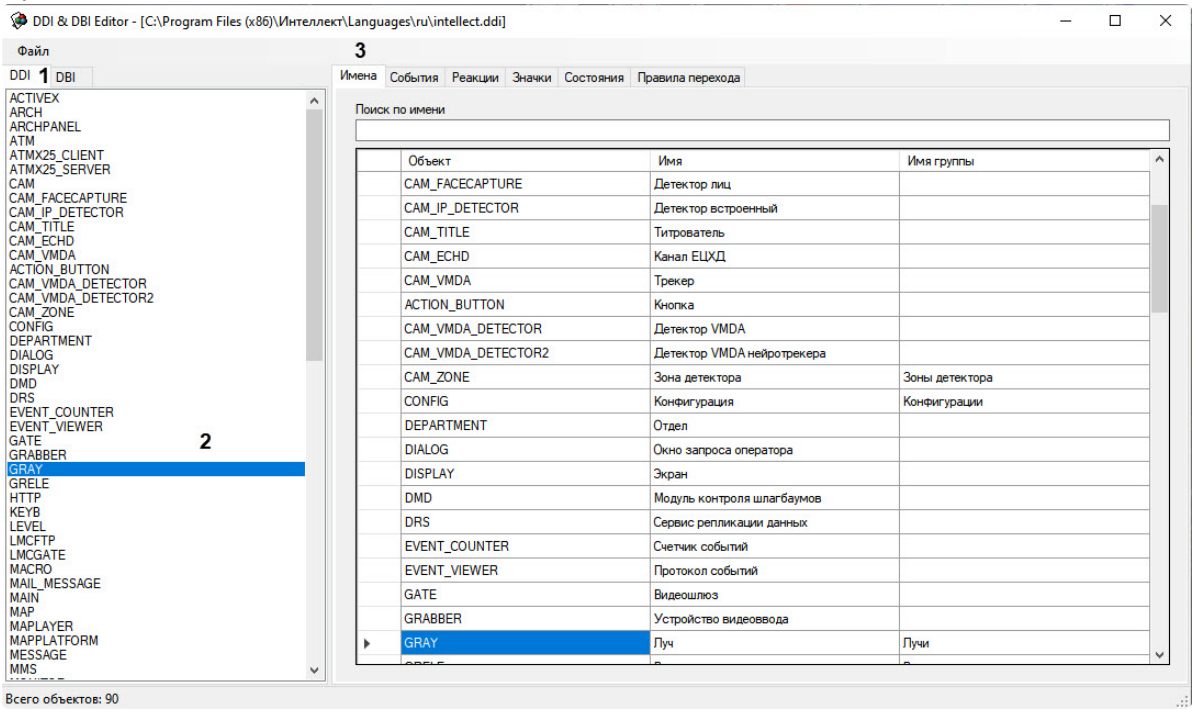
Утилита *ddi.exe* запускается одним из следующих способов:

1. Из меню **Пуск** → **Интеллект** → **Настройка конфигурации**.
2. Из папки **Tools** директории установки ПК *Интеллект*.

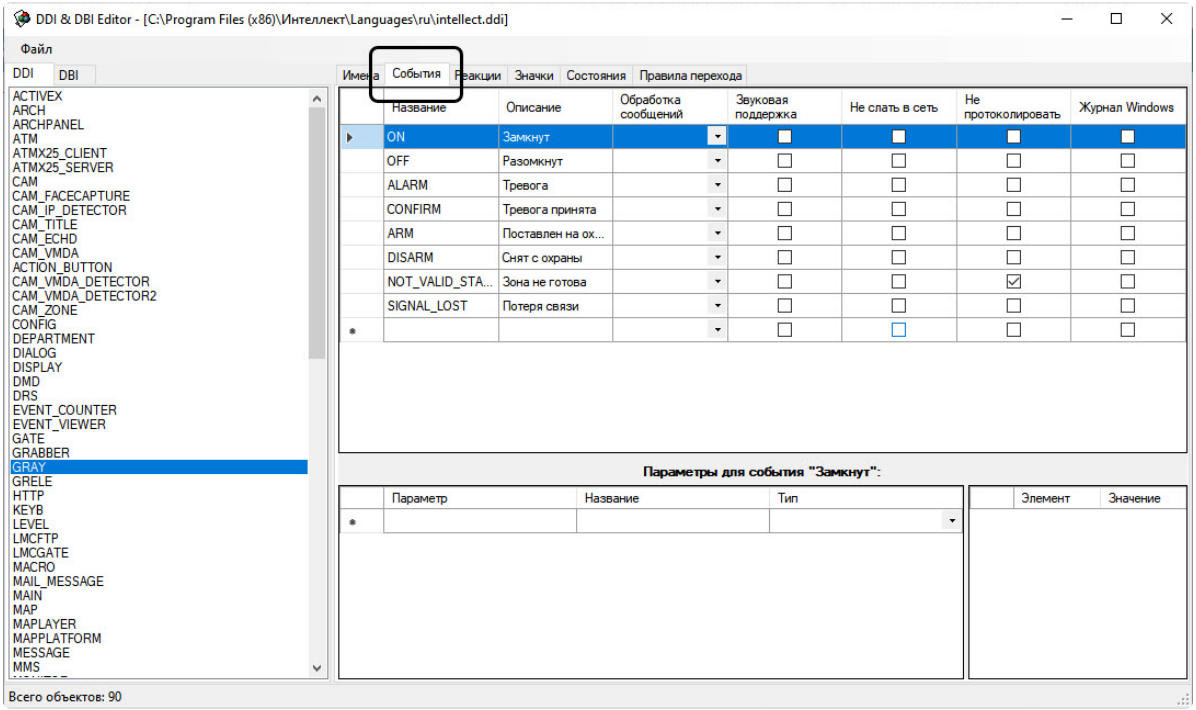
Для просмотра списка системных названий объектов, событий и реакций необходимо:

1. Открыть в утилите файл intellect.ddi.

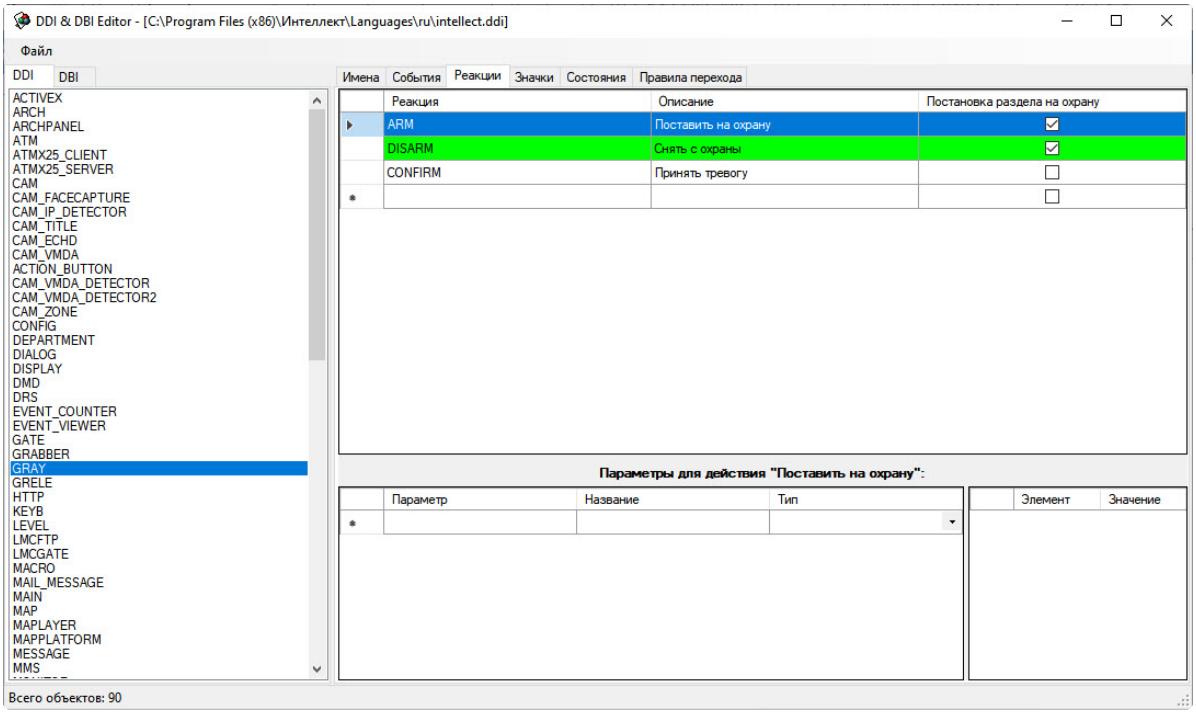
2. Выбрать вкладку **DDI** в левой части окна утилиты (**1**). В списке на данной вкладке представлен перечень системных объектов.



3. Выбрать в списке на вкладке **DDI** объект, события и реакции которого требуется просмотреть (**2**).
4. Для просмотра имени выбранного объекта необходимо перейти на вкладку **Имена** (**3**).
5. Для просмотра списка событий выбранного объекта необходимо перейти на вкладку **События**.



6. Для просмотра списка реакций выбранного объекта необходимо перейти на вкладку **Реакции**.



Более подробно работа с утилитой ddi.exe описана в разделе Редактирование шаблонов основной базы данных intellect.dbi и intellect.ext.dbi с помощью утилиты ddi.exe.

> **Примечание.**
>
> Если луч поставлен на охрану, то при замыкании/размыкании луча, в зависимости от настройки режима срабатывания приходит событие «Тревога» (см. Руководство по установке и настройке компонентов охранной системы, раздел Создание и настройка системного объекта Луч). Если луч снят с охраны, то приходят события «Замкнут"/"Разомкнут» соответственно.

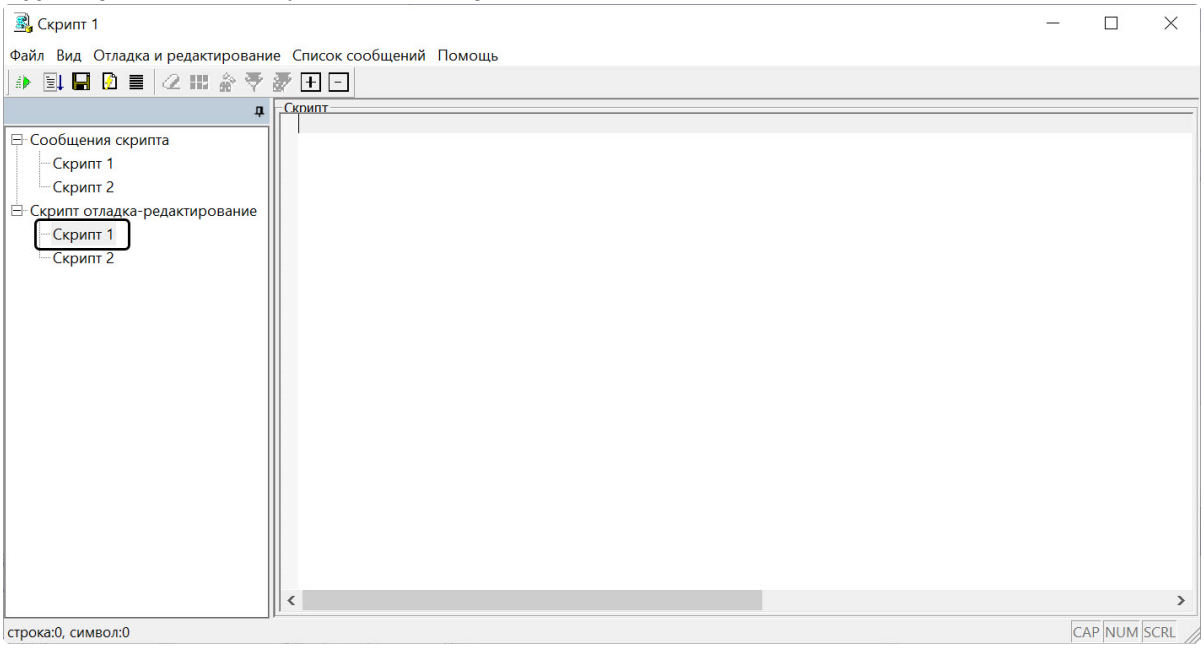
9.11 Создание первого скрипта

В качестве примера использования языка программирования JScript в программном комплексе *Интеллект* вначале предлагается создать скрипт, содержащий ошибку, а впоследствии внести в него исправления. Скрипт выполняет следующие действия: по запуску **Макрокоманды № 1** скрипт должен устанавливать для камер № 1-4 значение параметра **Горячая запись** равным 10 и выводит в отладочное окно утилиты *Редактор-Отладчик* сообщение «Hello world».

Для создания и запуска данного скрипта необходимо выполнить следующие действия:

1. Во вкладке **Оборудование** диалогового окна **Настройка системы** создать четыре объекта **Камера** с идентификационными номерами 1, 2, 3 и 4, если они не были созданы ранее.

2. Во вкладке **Программирование** создать объект **Макрокоманда** с идентификационным номером 1. Таблицу **События** заполнять не требуется для корректного выполнения последующих действий и успешного запуска скрипта.
3. Во вкладке **Программирование** создать системный объект **Скрипт.** Задать объекту идентификационный номер 1 и название "Скрипт 1".
4. В панели настройки системного объекта **Скрипт 1** из списка **Временная зона** выбрать пункт **Всегда.**
5. Нажать кнопку **Редактор-Отладчик**, расположенную в нижней части панели настройки системного объекта **Скрипт 1**. После выполнения указанного действия на экран будет выведено окно утилиты *Редактор-Отладчик*.
6. В окне утилиты *Редактор-Отладчик* необходимо раскрыть список **Скрипт отладка-редактирование** и выбрать объект **Скрипт 1**.

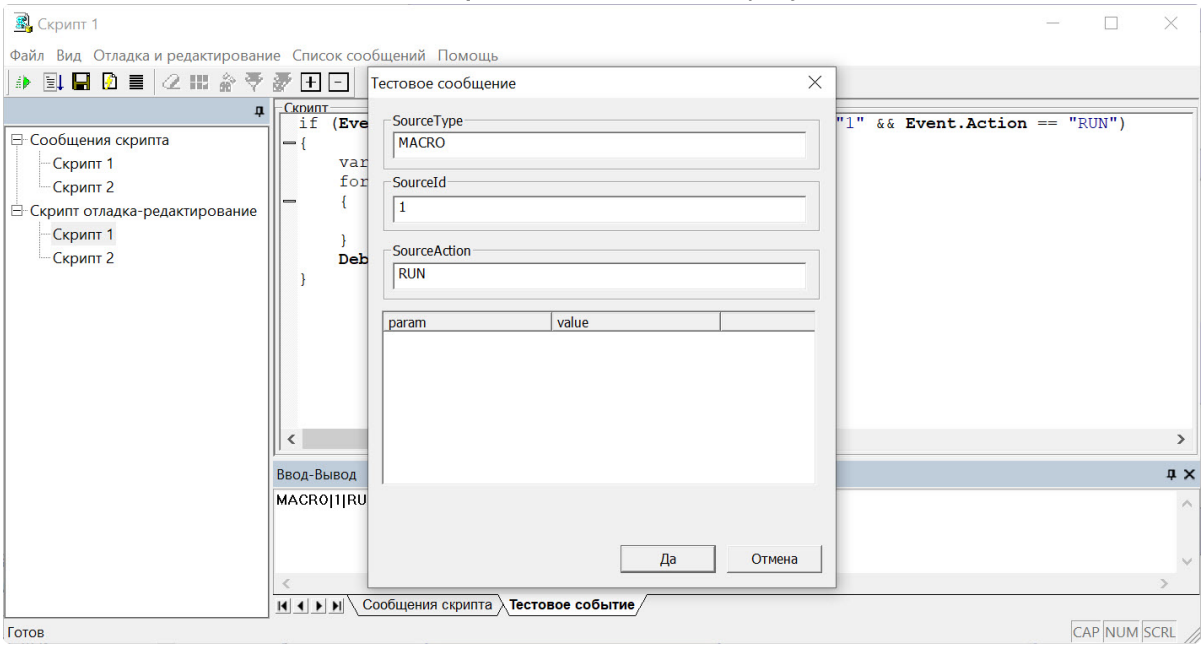


7. В поле **Скрипт** ввести следующие строки:

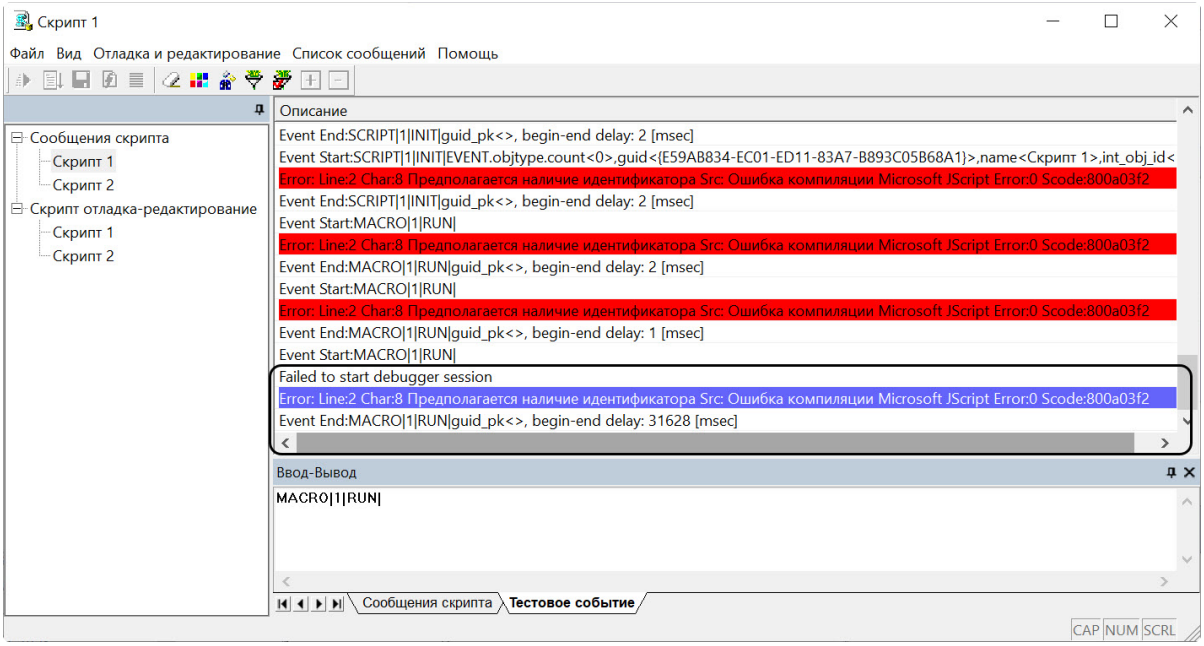
```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action ==
"RUN")
{
    var ;
    for(i=1;i<=4;i=i+1)
    {
        SetObjectParam("CAM",i,"hot_rec_time","10");
    }
    DebugLogString ("Hello world");
}
```

> **Внимание!**
>
> Скрипт содержит ошибку. Рекомендации по ее устранению приводятся ниже.

8. Сохранить скрипт, выбрав в меню **Файл** утилиты команду **Сохранить в базе**.
9. Создать тестовое событие для запуска скрипта в режиме отладки – MACRO|1|RUN|. Для этого необходимо выбрать в меню **Отладка и редактирование** команду **Редактировать тестовое событие**, при этом на экран будет выведено окно **Тестовое сообщение**. Необходимо заполнить поля окна **Тестовое сообщение**, как показано на рисунке.



10. Запустить скрипт по тестовому событию, выбрав в меню **Отладка и редактирование** команду **Тестовый Пуск**.
11. Раскрыть список **Сообщения скрипта** и выбрать пункт **Скрипт 1**. В правой части окна утилиты отобразится **Отладочное окно** скрипта.
12. В отладочном окне найти строку "Event Start:MACRO|1|RUN|" и сообщение об ошибке: "Предполагается наличие идентификатора Src: Ошибка компиляции Microsoft JScript Line:2 Char:8 Error:0 Scode:800a03f2".



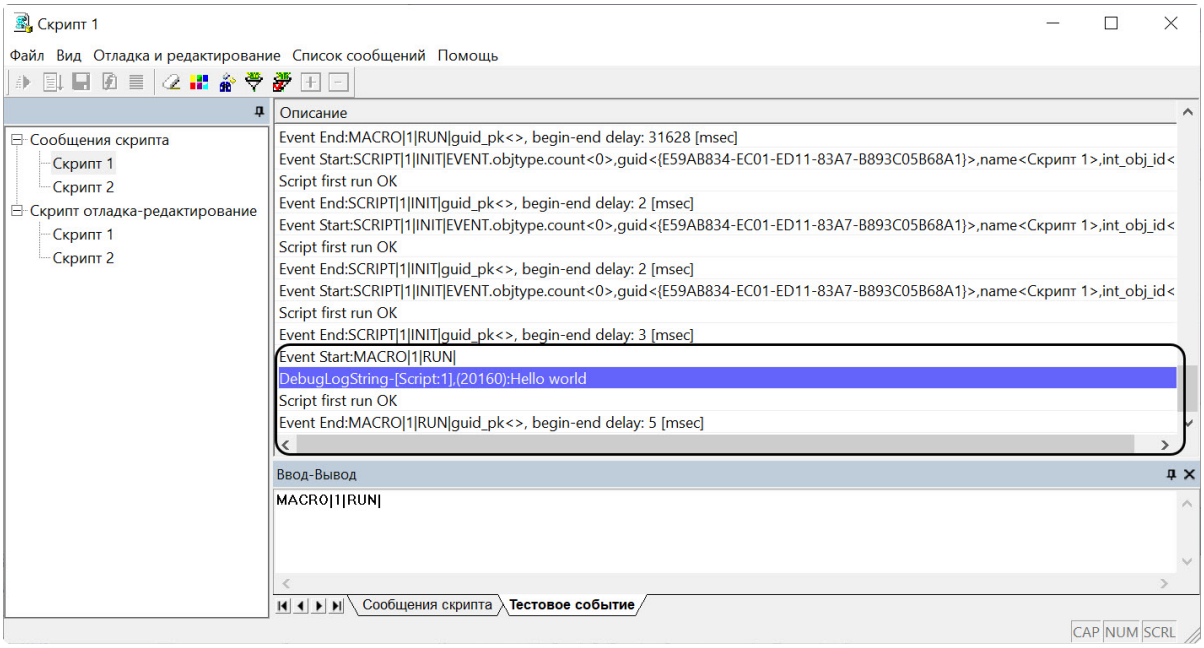
Сообщение об ошибке указывает, что в строке 2 данного скрипта в операторе объявления переменных (var) отсутствует идентификатор, то есть ни одна переменная объявлена не была. В соответствии с правилами языка JScript, это считается критической ошибкой, и выполнение скрипта не осуществляется.

13. Внести исправления в текст скрипта, как показано ниже (см. строку var i;).

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action ==
"RUN")
{
    var i;
    for(i=1; i<=4; i=i+1)
    {
        SetObjectParam("CAM",i,"hot_rec_time","10");
    }
    DebugLogString ("Hello world");
}
```

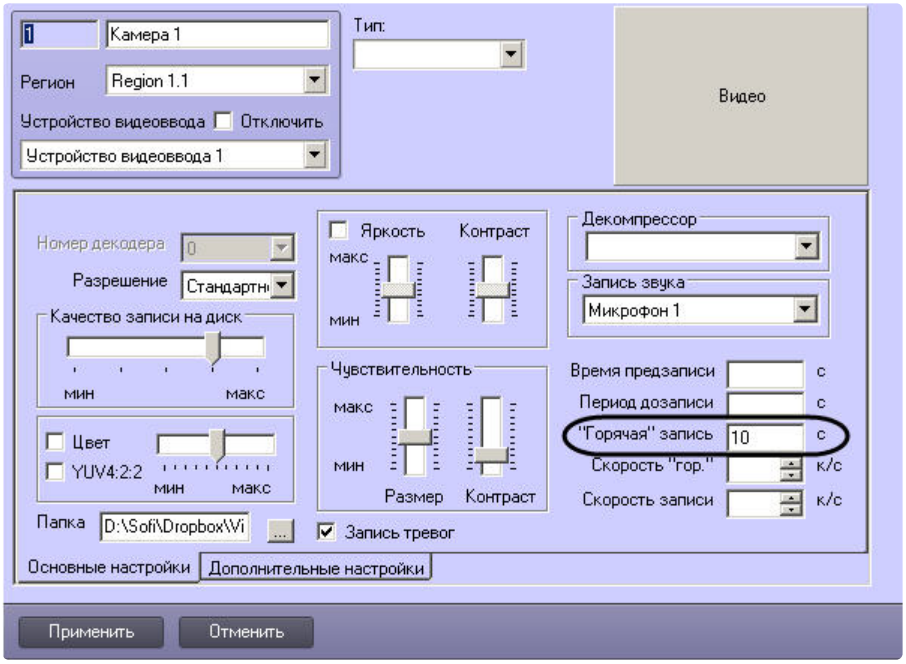
14. Сохранить скрипт, выбрав в меню **Файл** утилиты команду **Сохранить в базе**.
15. Повторить действия 10 и 11.
16. В отладочном окне найти строку "Event Start:MACRO|1|RUN|" и сообщения" DebugLogString:Hello world" и "Script first run OK". Сообщение "Script first run OK" свидетельствует о том, что скрипт

корректно работает в режиме отладки.



17. Завершить работу с утилитой *Редактор-Отладчик.*
18. В поле системного объекта **Скрипт 1** отобразится текст созданного скрипта. Для активирования скрипта в панели настройки системного объекта **Скрипт 1** требуется нажать кнопку **Применить**.
19. Вызвать из меню **Выполнить** Главной панели управления макрокоманду №1.
20. С помощью отладочного окна ПК *Интеллект* убедиться в успешном запуске макрокоманды и выполнении скрипта.

21. Убедиться в корректном выполнении скрипта. В панели настойки системных объектов **Камера № 1-4** в поле **Горячая запись** должно быть указано значение 10.



> **Примечание.**
>
> По умолчанию поля **Горячая запись** в панелях настройки объектов **Камера** не заполнены.

Процесс создания и отладки скрипта завершен.

9.12 Работа со скриптом

9.12.1 Создание скрипта

На странице:

Для создания и запуска скриптов на языке JScript необходимо создать системный объект **Скрипт**, затем в утилите *Редактор-Отладчик* ввести скрипт, проверить его и отладить.

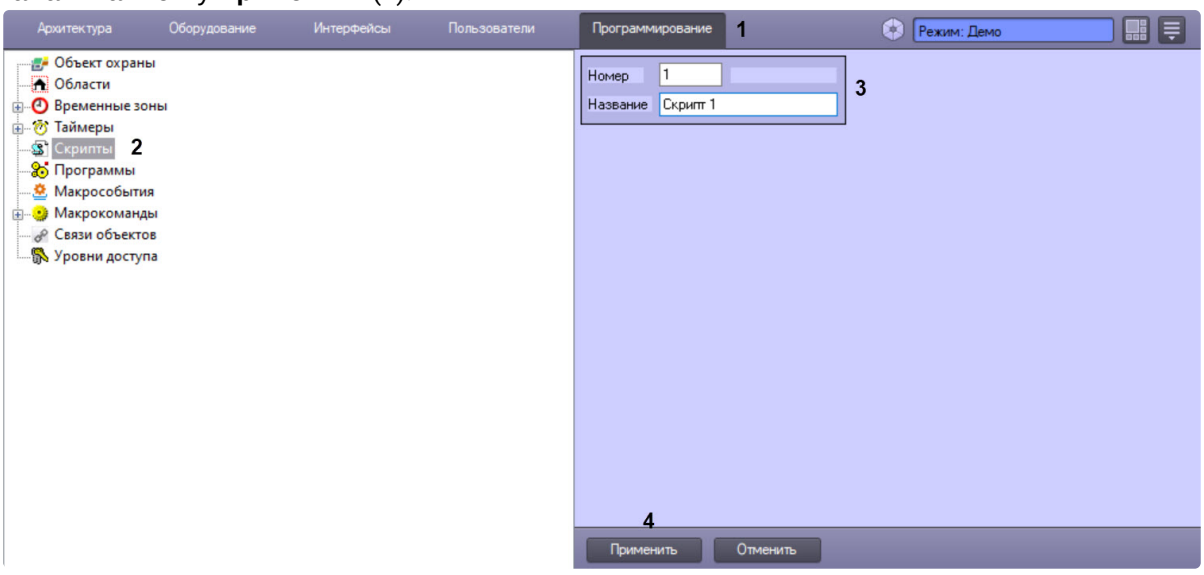
Создание скриптов на языке JScript в программном комплексе *Интеллект* осуществляется с использованием встроенной утилиты *Редактор-Отладчик*.

Утилита *Редактор-Отладчик* запускается с помощью кнопки **Редактор-Отладчик**, расположенной на панели настройки системного объекта **Скрипт.**

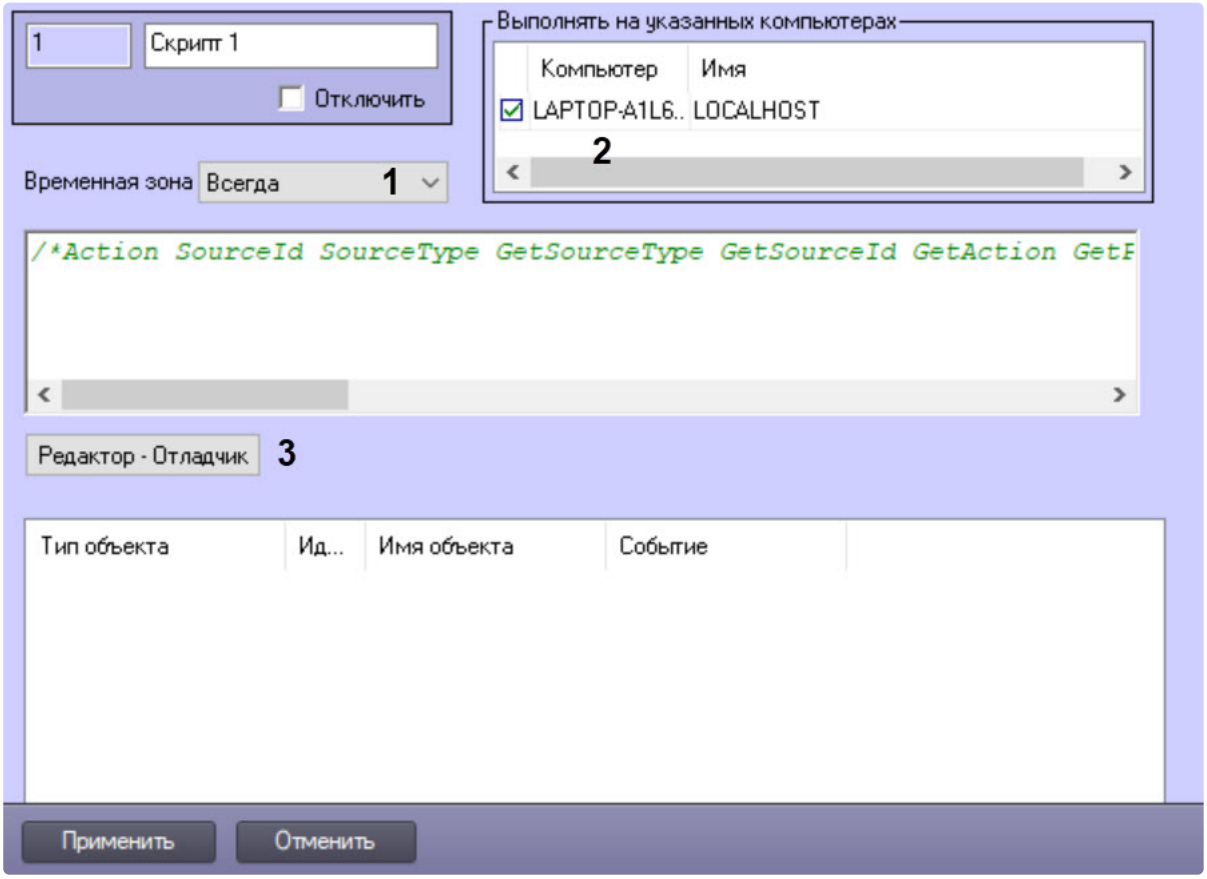
9.12.1.1 Создание объекта Скрипт

Для создания объекта **Скрипт** в программном комплексе *Интеллект* необходимо выполнить следующие действия:

1. На вкладке **Программирование** (**1**) диалогового окна **Настройка системы** создать объект **Скрипт** (**2**).
2. Задать объекту **Скрипт** идентификационный номер и название (**3**).
3. Нажать на кнопку **Применить** (**4**).



В результате появится панель настройки объекта **Скрипт**.



Для настройки объекта **Скрипт** необходимо задать значения следующим параметрам:

1. В поле **Временная зона** требуется указать временную зону выполнения скрипта: **Всегда**, **Никогда** или одна из зон, созданных ранее (**1**, см. Создание и настройка временных зон). По умолчанию **Никогда**.
2. В панели **Компьютеры** указать компьютеры (ядра), на которых требуется выполнять создаваемый скрипт (**2**).

> **Примечание.**
>
> По умолчанию скрипт настроен на выполнение на всех компьютерах (ядрах). В списке компьютеров отображаются только те компьютеры, которые зарегистрированы на вкладке **Оборудование** диалогового окна **Настройка системы**.

3. Нажать **Применить**.

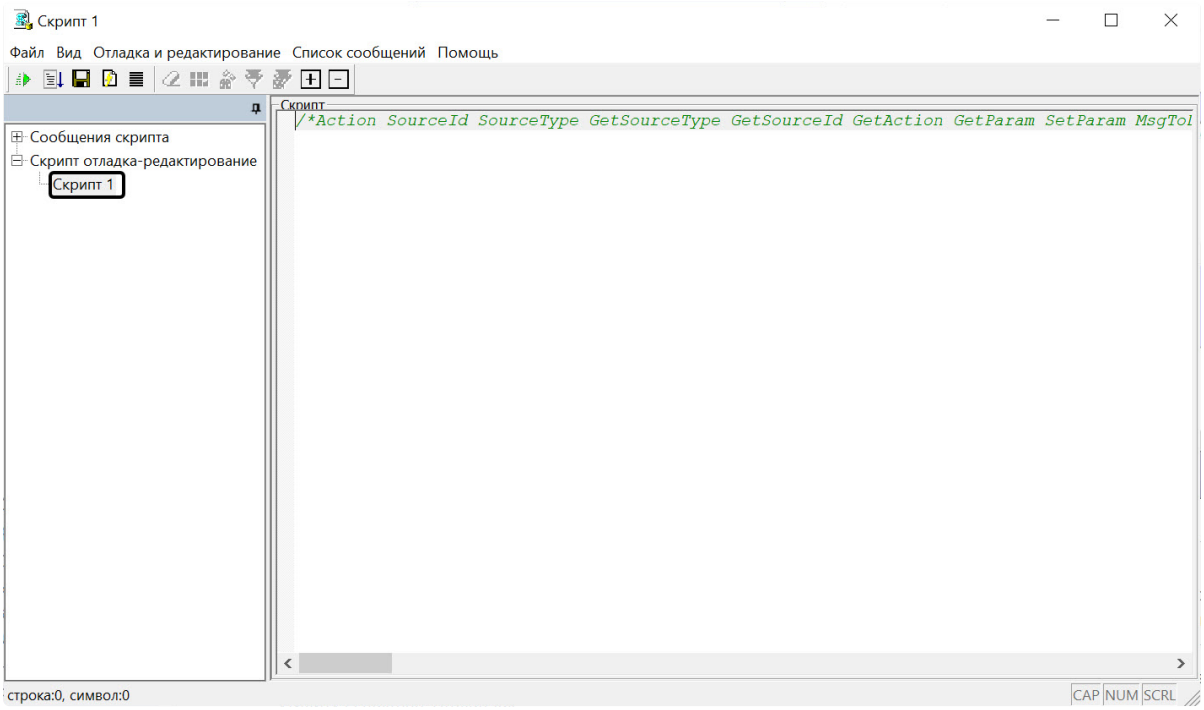
9.12.1.2 Создание и редактирование скрипта

Для создания скрипта на языке программирования JScript в программном комплексе *Интеллект* необходимо:

1. С помощью кнопки **Редактор-Отладчик**, расположенной в нижней части панели системного объекта **Скрипт**, вызвать утилиту *Редактор-Отладчик* (**3**).

> **Примечание.**
>
> Создание, редактирование и сохранение скрипта на языке программирования JScript рекомендуется осуществлять с помощью утилиты *Редактор-Отладчик*. На панели настройки системного объекта **Скрипт** отображается текст сохраненного скрипта, также доступный для редактирования.

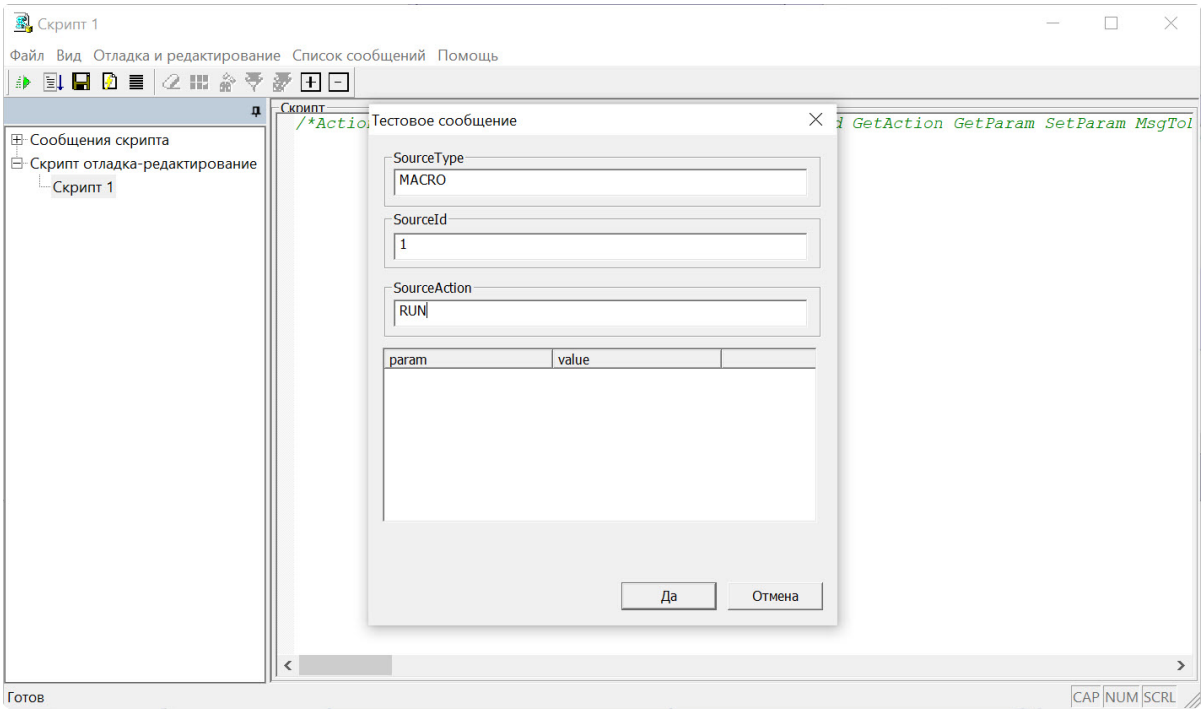
2. В окне утилиты *Редактор-Отладчик* необходимо раскрыть список **Скрипт отладка-редактирование** и выбрать объект **Скрипт**, редактирование которого необходимо выполнить (например, объект **Скрипт 1**, как на рисунке ниже).



3. В поле **Скрипт** необходимо ввести текст скрипта на языке программирования JScript (см. Примеры скриптов на языке JScript).

4. Запустить скрипт, воспользовавшись тестовым событием. Для создания тестового события воспользоваться командой **Отладка и редактирование → Редактировать тестовое событие.** В результате на экран будет выведено окно **Тестовое сообщение**, содержащее поля для

задания параметров события.



Для запуска скрипта по тестовому событию необходимо воспользоваться командой **Отладка и редактирование → Тестовый пуск**.

> **Примечание**
>
> Запуск скрипта по тестовому событию также возможен с помощью сочетания клавиш **Ctrl + T**.

9.12.1.2.1 Возможности работы со скриптом

При работе со скриптом в утилите *Редактор-Отладчик* имеется возможность отменить последнее действие, а также вернуть последнее действие. Для отмены последнего действия нужно нажать сочетание клавиш **Ctrl+Z**, для возврата последнего действия – **Ctrl+Y.**

> **Примечание.**
>
> Для удобства редактирования скрипта позиция курсора запоминается при сохранении скрипта и при переходе между скриптами в окне утилиты *Редактор-Отладчик* в течение одной сессии, т.е. до перезапуска ПК *Интеллект*.

Примечание.

При переходе к списку **Сообщения скрипта** во время редактирования скрипта, а затем обратно к соответствующем скрипту в списке **Скрипт отладка-редактирование,** отменить или вернуть последнее действие невозможно.

9.12.1.3 Отладка скрипта

Проверка скрипта на корректность синтаксиса выполняется встроенным в утилиту *Редактор-Отладчик* интерпретатором. Результат проверки с информацией о содержании и местонахождении ошибки отображается в соответствующем скрипту **Отладочном окне** в списке **Сообщения скрипта**. При наличии ошибок необходимо внести правки в синтаксис скрипта и повторить проверку.

Примечание.

Подробная информация об использовании тестовых событий для отладки скриптов приведена в главе Отладка скриптов.

После отладки скрипта средствами утилиты *Редактор-Отладчик* запустить его по реальному системному событию. Проверить результат выполнения скрипта. В случае некорректности выполнения скрипта внести необходимые изменения и повторно запустить скрипт.

Процесс создания скрипта считается завершенным в том случае, если скрипт выполняется корректно.

9.12.2 Сохранение скрипта

Утилита *Редатор-Отладчик* обеспечивает два способа сохранения скриптов: в системном объекте **Скрипт** или в текстовом файле на диске компьютера.

Сохранение скрипта в системном объекте **Скрипт** осуществляется по команде **Файл → Сохранить в базе**. При выборе данного пункта меню скрипт будет обновлен на всех Серверах, которые выбраны на панели настройки объекта **Скрипт** (см. Системный объект Скрипт).

Примечание

Сохранение скрипта в базе также возможно с помощью сочетания клавиш **Ctrl+S**.

Примечание.

> Скрипт автоматически сохраняется в соответствующем ему системном объекте **Скрипт** при завершении работы с утилитой *Редактор-Отладчик*.

Сохранение скрипта в файл выполняется по команде **Файл → Сохранить на диск**. Сохраненный в файл скрипт впоследствии может быть загружен в утилиту *Редактор-Отладчик* с помощью команды **Файл → Загрузить с диска**.

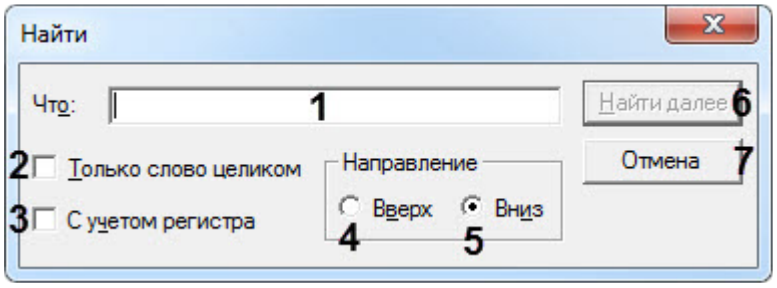
9.12.3 Удаление скрипта

Удаление созданного в программном комплексе *Интеллект* скрипта осуществляется путем удаления соответствующего ему системного объекта **Скрипт**, размещенного во вкладке **Программирование**.

9.12.4 Поиск в скрипте

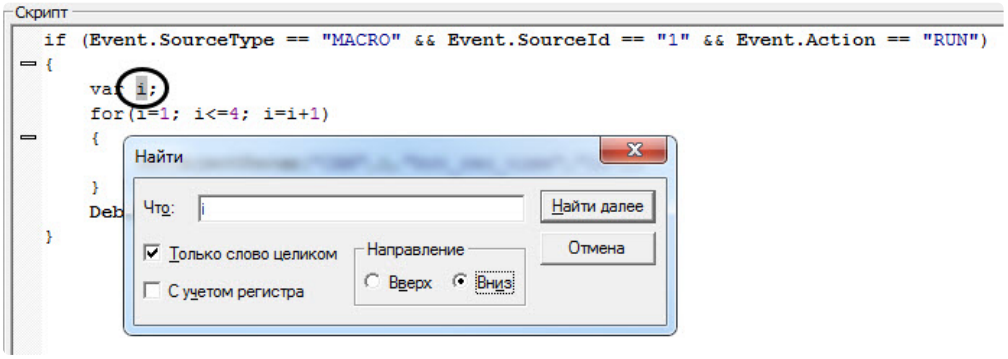
В утилите *Редатор-Отладчик* существует возможность открытия диалогового окна поиска текста в скрипте.

Для того, чтобы открыть диалоговое окно поиска, необходимо нажать сочетание клавиш **Ctr l+ F**. Откроется окно **Найти**.



1. В поле (**1**) необходимо ввести текст, который нужно найти в скрипте.
2. Установить флажок **Только слово целиком** (**2**), если необходимо искать введенный текст целиком.
3. Установить флажок **С учетом регистра** (**3**), если необходимо искать введенный текст с учетом регистра.
4. Выбрать направление поиска по скрипту относительно текущего положения курсора: **Вверх** (**4**) или **Вниз** (**5**).
5. Нажать на кнопку **Найти и далее** (**6**) для начала поиска и перехода к следующему совпадению.
6. Для отмены поиска нажать на кнопку **Отмена** (**7**).

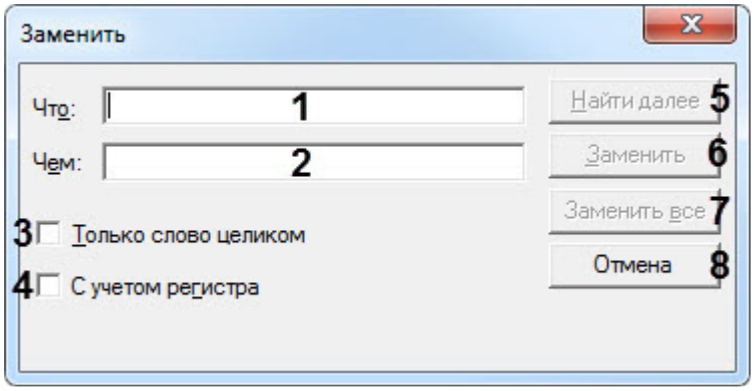
При нахождении совпадения найденный текст выделится в окне утилиты *Редатор-Отладчик*.



9.12.5 Замена в скрипте

В утилите *Редатор-Отладчик* существует возможность открытия диалогового окна замены текста в скрипте.

Для того, чтобы открыть диалоговое окно замены, необходимо нажать сочетание клавиш **Ctrl+H**. Откроется окно **Заменить**.



1. В поле **Что** (**1**) необходимо ввести текст, который нужно найти в скрипте.
2. В поле **Чем** (**2**) необходимо ввести текст, которым нужно заменить найденный текст в скрипте.
3. Установить флажок **Только слово целиком** (**3**), если необходимо искать введенный текст целиком.
4. Установить флажок **С учетом регистра** (**4**), если необходимо искать введенный с учетом регистра введенного текста.
5. Нажать на кнопку **Найти и далее** (**5**) для начала поиска и перехода к следующему совпадению.

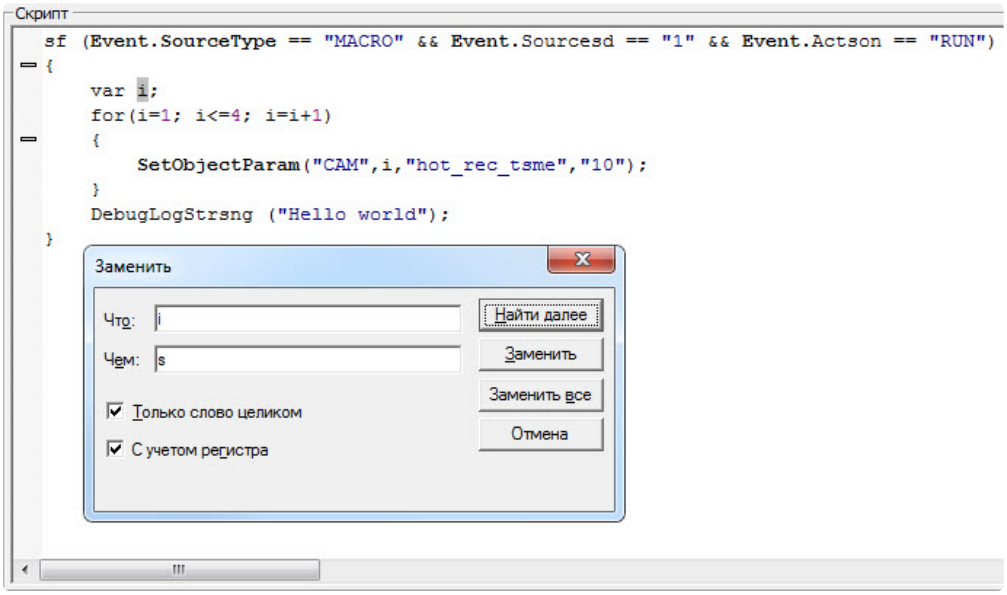
> **Примечание**
>
> Поиск по скрипту происходит вниз относительно текущего положения курсора.

6. Нажать на кнопку **Заменить** (**6**) чтобы заменить текущее найденное совпадение.

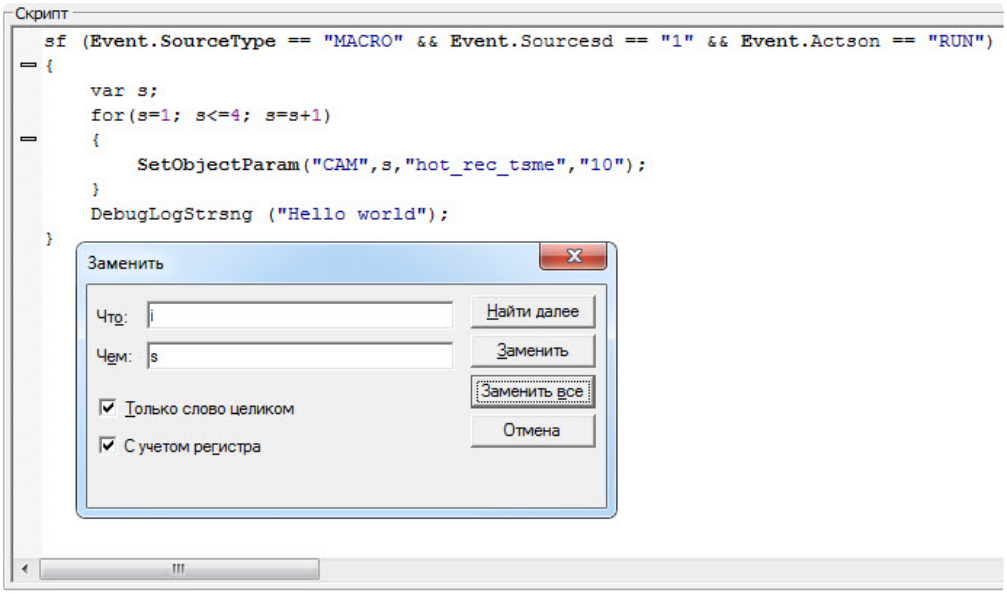
7. Нажать на кнопку **Заменить все** (**7**) чтобы заменить все совпадения автоматически.
8. Для закрытия диалогового окна замены нажать на кнопку **Отмена** (**8**).

Пример использования замены переменной **i** на **s**:

1. До замены:



2. После замены:



9.13 Отладка скриптов

9.13.1 Возможности отладки скриптов

Утилита *Редактор-Отладчик* обеспечивает отладку скриптов с помощью встроенных программных средств проверки корректности синтаксиса, интерпретации скриптов, запуска по тестовым событиям, генерируемым утилитой. Процесс отладки сопровождается отображением сообщений о результатах проверки и выполнения скрипта в отладочных окнах.

Возможности *Редактор-Отладчика*:

1. Каждому системному объекту **Скрипт** соответствует отдельное отладочное окно, в которое выводятся системные и тестовые события, сообщения об ошибках и успешном выполнении скриптов, а также пользовательские информационные сообщения. Предусмотрено использование фильтров вывода сообщений в отладочных окнах.
2. Предусмотрено использование специализированных отладочных окон **Информационное окно**, в которых отображаются сообщения, относящиеся только к отлаживаемому скрипту.
3. Для проверки работоспособности скрипта могут быть использованы тестовые события, генерируемые утилитой *Редактор-Отладчик* (тестовые события не регистрируются в системе).
4. Допускается использование сторонних программ-отладчиков, реализующих функции пошагового выполнения скриптов (Step), просмотра значений переменных скриптов в процессе их выполнения (Watch) и др.

9.13.2 Создание и использование тестовых событий

На странице:

9.13.2.1 Создание тестовых событий

Для удобства отладки скриптов в утилите *Редактор-Отладчик* реализована возможность использования тестовых событий, задаваемых пользователем и генерируемых утилитой. Тестовые

события не регистрируются на уровне системы видеонаблюдения: не отображаются в протоколе событий и не записываются в базу данных.

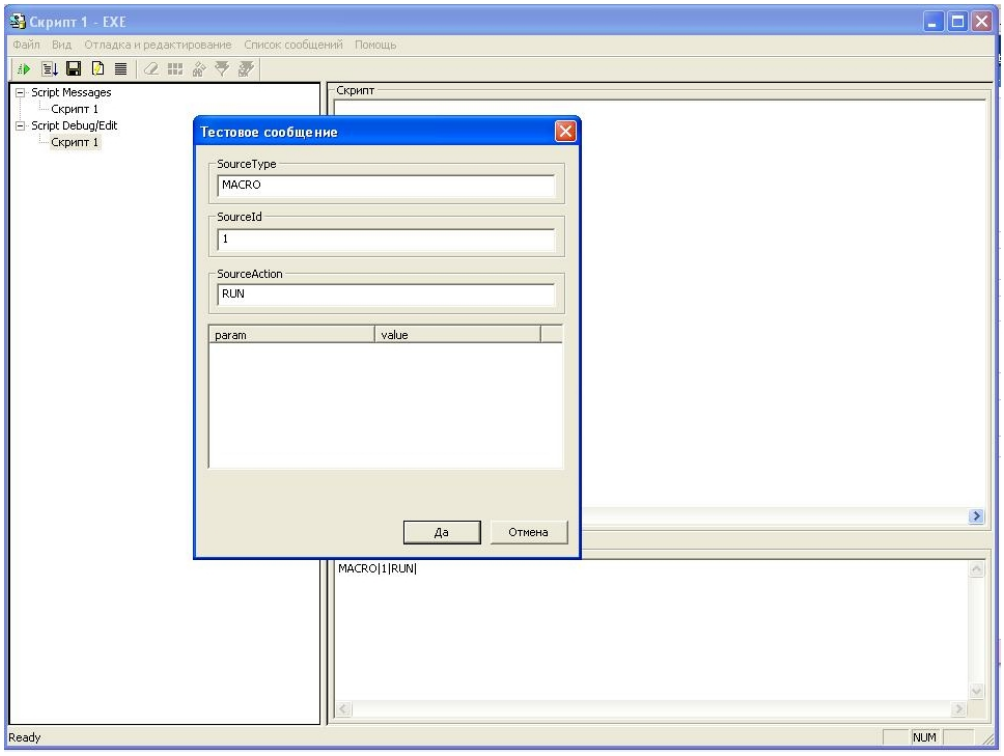
Для каждого скрипта допускается создание не более одного тестового события.

Для создания тестового события выполнить действия:

1. Выбрать в меню **Отладка и редактирование** команду **Редактировать тестовое событие**.

   Данную команду также можно вызвать с панели инструментов нажатием кнопки  .

2. На экран будет выведено окно **Тестовое сообщение**. Оно предназначено для ввода параметров тестового события.



3. В полях окнах **Тестовое сообщение** ввести:
   a. **SourceType** − тип системного объекта;
   b. **SourseId** − идентификационный номер системного объекта;
   c. **SourceAction** − событие, генерируемое заданным системным объектом;
   d. **param** − дополнительные параметры события;
   e. **value** − значения дополнительных параметров.

> **Примечания**

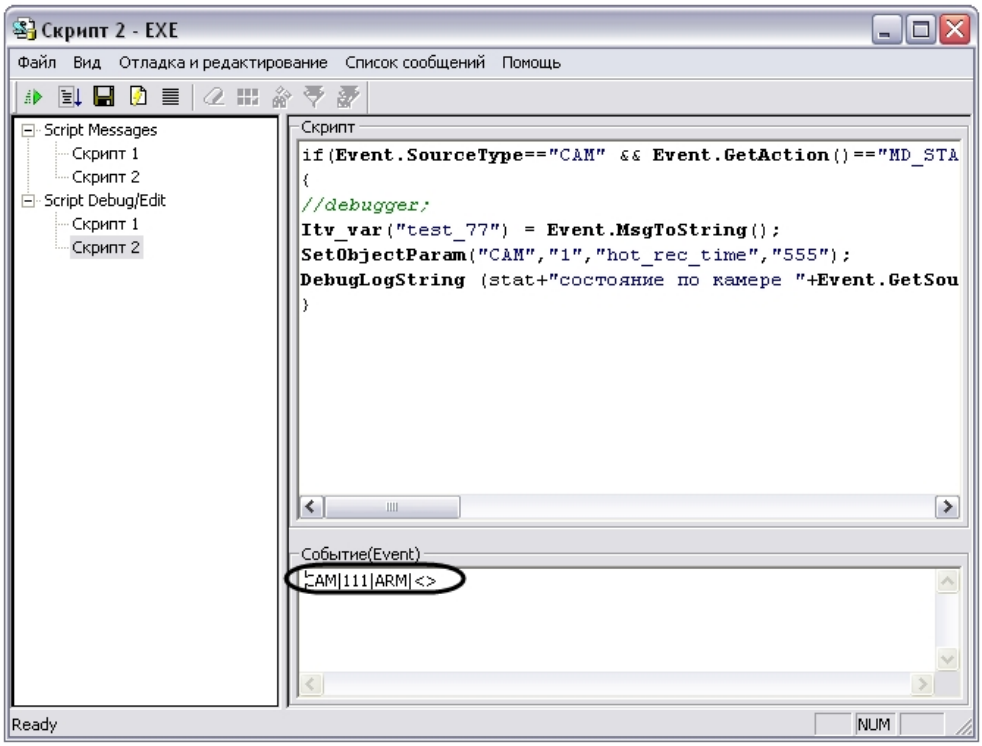
> Основные объекты системы, их события и параметры описаны в разделе Описание событий и реакций объектов системы. Также их можно получить с помощью утилиты ddi.exe (см. Получение списка системных названий объектов, реакций и событий ПК Интеллект).

4. По окончании заполнения полей окна **Тестовое сообщение** необходимо нажать кнопку **Да.**

Процесс создания тестового события завершен.

После создания тестовое событие отобразится в поле **Событие(Event)** в специализированном строковом формате.

Например, на рисунке ниже в качестве тестового события указано **Постановка на охрану камеры № 111**.



9.13.2.2 Запуск скрипта по тестовому событию

Запустить скрипт по тестовому событию можно одним из способов:

1. Нажать кнопку **Тестовый запуск** , расположенную на панели инструментов утилиты.
2. Выбрать в меню **Отладка и редактирование** команду **Тестовый Пуск**.
3. Выбрать в меню **Отладка и редактирование** команду **Тестовый пуск с Отладчиком**.

При запуске скрипта по команде **Тестовый пуск с Отладчиком** осуществляется запуск сторонней программы-отладчика (см. Использование сторонних программ-отладчиков).

Сообщения о результатах проверки и выполнения скрипта отображаются в соответствующем скрипту отладочном окне утилиты *Редактор-Отладчик*.

9.13.3 Работа с отладочными окнами утилиты Редактор-Отладчик

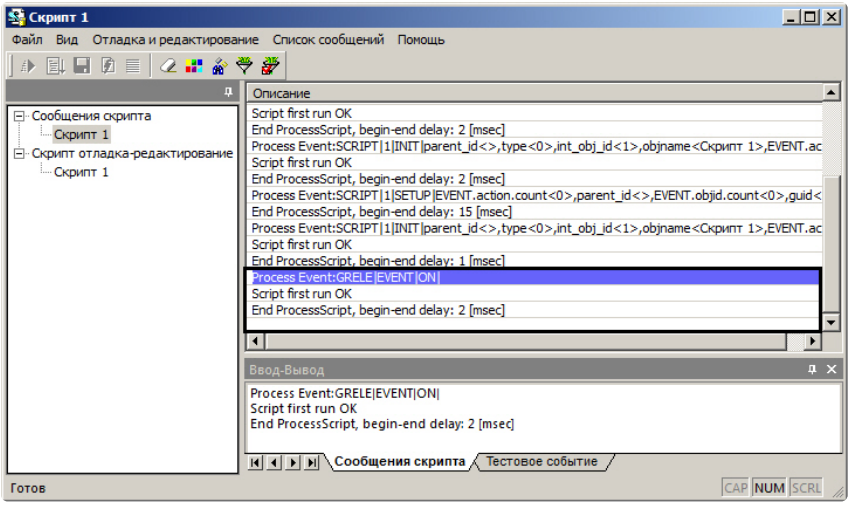
9.13.3.1 Просмотр сообщений скрипта

Отладочные окна предназначены для отображения сообщений о регистрации системных и тестовых событий, об ошибках и успешном выполнении скриптов, а также пользовательских информационных сообщений.

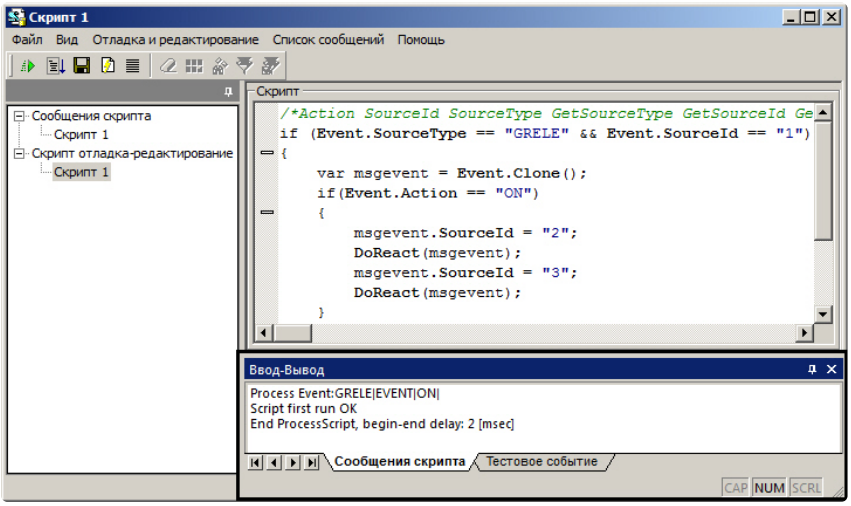
Для каждого скрипта в утилите *Редактор-Отладчик* предусмотрено отдельное отладочное окно.

Существует два типа отладочных окон: **Все сообщения скрипта** и **Сообщения скрипта при последнем запуске**.

Отладочные окна типа **Все сообщения скрипта** размещаются в списке **Сообщения скрипта**. Названия отладочных окон совпадают с названиями соответствующих им объектов **Скрипт**. В данные окна выводятся все системные сообщения, относящиеся к соответствующему скрипту. Пример отладочного окна типа **Все сообщения скрипта**:



Кроме того, информация о последнем запуске скрипта отображается на вкладке **Сообщения скрипта** на панели в нижней части окна утилиты *Редактор-отладчик*. Если эта панель не отображается, следует выбрать **Отладка и редактирование** → **Сводная информация**, либо нажать кнопку  на панели инструментов.



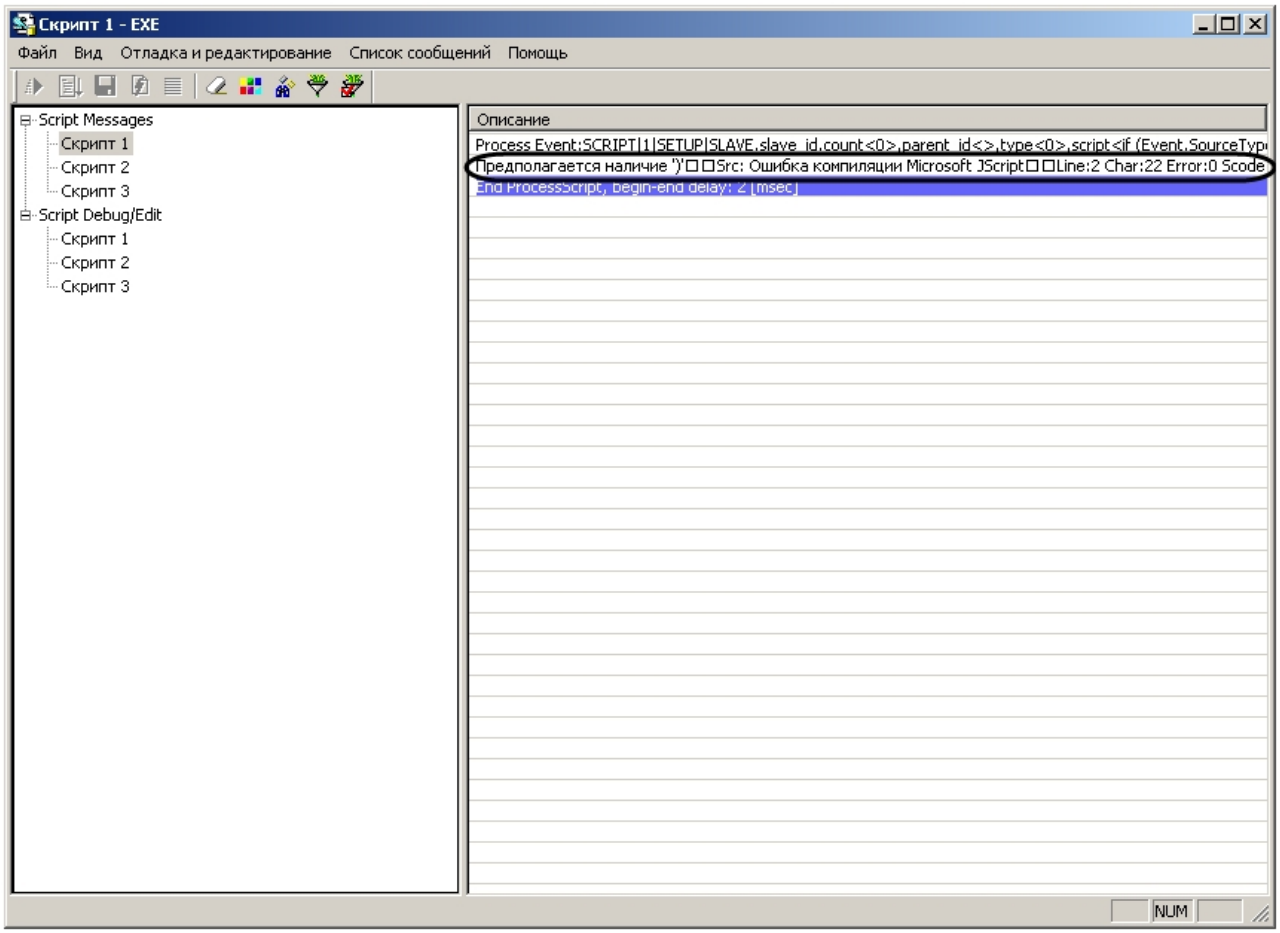
Порядок работы со всеми отладочными окнами любых типов одинаков.

9.13.3.2 Отображение сообщений о запуске, проверке, изменении и выполнении скриптов в отладочных окнах

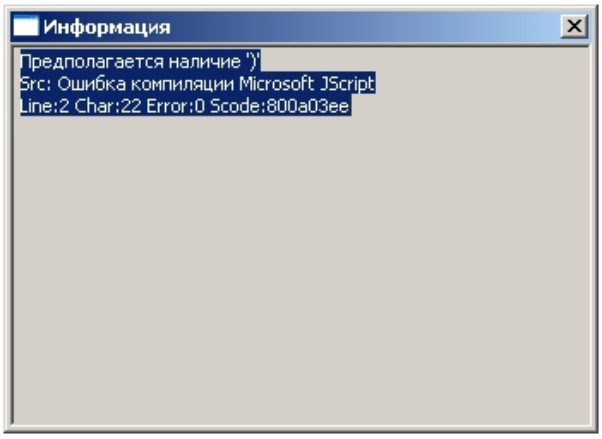
В отладочные окна последовательно выводятся сообщения, соответствующие этапам запуска, проверки и выполнения скриптов.

В момент регистрации события, по которому должен запуститься скрипт, в отладочное окно выводится сообщение «Process Event: событие, инициировавшее запуск скрипта» (например, при запуске скрипта по макрокоманде №1 в отладочное окно выводится строка «Process Event:MACRO|1|RUN|»). В момент изменения скрипта в утилите *Редактор-Отладчик* или в ПК *Интеллект* в отладочное окно выводится сообщение «Process Event:SCRIPT|номер скрипта|SETUP|» (например, при изменении скрипта с номером 1, в отладочное окно выводится строка «Process Event:SCRIPT|1|SETUP|»).

Перед запуском скрипта проверяется его синтаксис. Если в процессе проверки синтаксиса скрипта были обнаружены ошибки, в отладочное окно будут выведены соответствующие сообщения. На рисунке приведен пример отображения в отладочном окне сообщения об ошибке при проверке синтаксиса скрипта.



Для просмотра полного содержания строки с сообщением нужно кликнуть правой клавишей мыши по данной строке. В результате на экран будет выведено окно **Информация**, в котором строка отобразится полностью.



Строка с сообщением об ошибке содержит информацию:

1. содержание ошибки;
2. тип ошибки (например, «Src: Ошибка компиляции Microsoft JScript»);

3. местоположение ошибки в тексте скрипта (номер строки «Line» и номер символа в строке «Char»);
4. код ошибки («Scode»).

Если при проверке синтаксиса скрипта не были обнаружены ошибки, в отладочное окно будет выведено сообщение: «Script first run OK». Далее запустится скрипт.

Если ошибки регистрируются в процессе выполнения скрипта, сообщения о них также выводятся в отладочном окне.

При успешном окончании выполнения скрипта в окно будет выведено сообщение «End ProcessScript, begin-end delay: время выполнения скрипта» (например, «End ProcessScript, begin-end delay: 13 [msec]»).

9.13.4 Использование сторонних программ-отладчиков

Программным комплексом *Интеллект* официально поддерживается отладчик Microsoft Visual Studio 2005.

ПК *Интеллект* допускает возможность использования сторонних программ для отладки скриптов на языке JScript. Сторонние программы-отладчики могут обеспечивать функциональные возможности отладки, не предусмотренные утилитой *Редактор-Отладчик*, например, пошаговое выполнение скриптов (функции "Step"), просмотр значений заданных в скриптах переменных в процессе выполнения скриптов (функция "Watch") и др.

> **Примечание.**
>
> Сторонние программы-отладчики следует использовать с осторожностью, поскольку они не обеспечивают полную совместимость с программным комплексом *Интеллект*. Необходимо учитывать, что использование сторонних программ-отладчиков может привести к аварийному завершению работы ПК *Интеллект*.

В случае использования для отладки скриптов сторонних программ-отладчиков настоятельно рекомендуется предварительно вводить в скрипт точку останова (Breakpoint). Ввод точки останова выполняется путем добавления в скрипт команды debugger;. При этом выполнение скрипта будет приостановлено в указанном командой debugger; месте и автоматически запустится программа-отладчик.

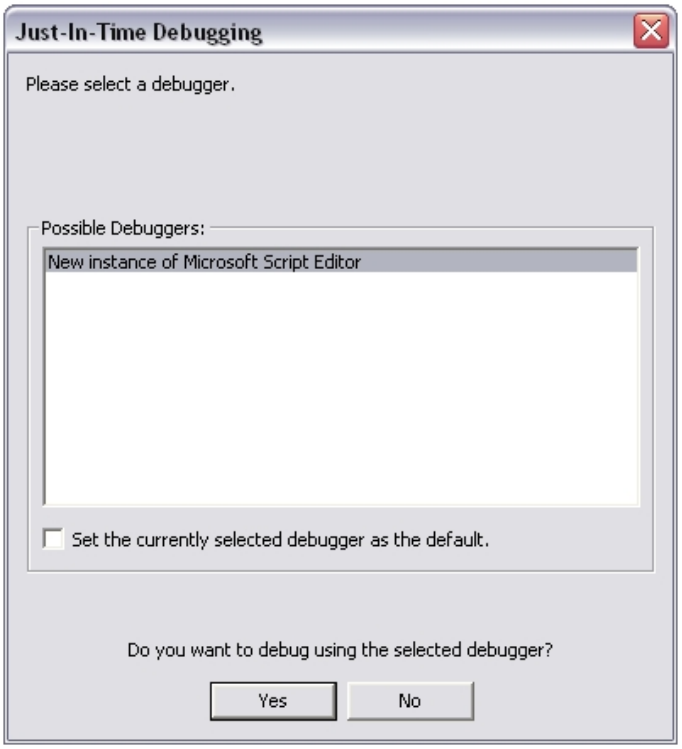
> **Примечание.**
>
> В программировании точкой останова (Breakpoint) называют преднамеренное прерывание выполнения программы, при котором выполняется вызов отладчика.

Запуск скрипта со сторонней программой-отладчиком можно выполнять только по тестовому событию.

Для запуска скрипта с отладкой в сторонней программе-отладчике нужно:

1. Разработать скрипт и добавить в него команду debugger;.

2. Создать тестовое событие для запуска скрипта.
3. Выбрать в меню **Отладка и редактирование** команду **Тестовый пуск с Отладчиком**.
4. На экран будет выведено диалоговое окно **Just In Time Debugging**. В данном окне из списка программ-отладчиков, установленных на компьютере, требуется выбрать необходимую программу.



5. Подтвердить выбор программы-отладчика нажатием кнопки **Yes**.
6. Если скрипт успешно пройдет проверку синтаксиса и до точки останова (команды debugger;) не будут обнаружены ошибки выполнения скрипта, будет произведен запуск сторонней программы-отладчика. При этом выполнение скрипта будет приостановлено в указанном точкой останова месте.

Пример. Скрипт с использованием точки останова после запуска макрокоманды №1.

```
if (Event.SourceType== "MACRO" && Event.SourceId=="1" && Event.Action == "RUN"); //
запуск макрокоманды №1
{
    debugger; //точка останова
    DebugLogString ("Hello world");
}
```

9.14 Примеры скриптов на языке JScript

Для иллюстрации возможных областей применения скриптов на языке JScript ниже приведены примеры, которые могут использоваться для создания на основе объекта **Скрипт** дополнительных функций в системе.

9.14.1 Примеры скриптов с Монитором видеонаблюдения и Камерами

MONITOR Монитор видеонаблюдения

CAM Камера

9.14.1.1 Пример 1. Визуализация работы детектора длины очереди в окне Монитора видеонаблюдения

Для корректной работы скрипта в ПК *Интеллект* предварительно должны быть созданы и настроены объекты **Детектор длины очереди** (входит в состав пакета детекторов Detector Pack), **Камера** и **Титрователь** (ниже вместо символов N, M, L нужно подставить соответствующие номера Детектора длины очереди, Камеры и Титрователя).

```
//Считывание события по длине очереди
if (Event.SourceType == "OCCUPANCY_COUNTER" && Event.SourceId == "N" && Event.Action
== "OCCUPANCY")  //N - Номер Детектора длины очереди
{
  var n=Event.GetParam("occupancy");
//Отображение длины очереди через Титрователь в Мониторе
  DoReactStr("CAM","M","CLEAR_SUBTITLES","title_id<L>"); //M - Номер Камеры  L -
Номер Титрователя
  DoReactStr("CAM","M","ADD_SUBTITLES","command<Длина очереди: "+n+" чел.
\r>,page<BEGIN>,title_id<L>"); //M, L - то же
}
```

В результате при отображении соответствующей камеры в Мониторе на видеоизображение будет накладываться текстовое сообщение о текущей длине очереди.

Настройки шрифта, цвета и положения надписи настраиваются на панели настройки объекта **Титрователь**.

Примечание

При использовании параметров page<BEGIN> и page<END> будут заполняться соответствующие поля в базе титров, что даст возможность производить поиск данных с помощью интерфейсного объекта **Поиск по титрам**.

9.14.1.2 Пример 2. Визуализация работы детектора подсчета посетителей в окне Монитора видеонаблюдения

Для корректной работы скрипта в ПК *Интеллект* предварительно должны быть созданы и настроены объекты **Детектор подсчета посетителей** (входит в состав пакета детекторов Detector Pack), **Камера**, **Титрователь** и **Макрокоманда** (ниже вместо символов N, M, L, P нужно подставить соответствующие номера Детектора подсчета посетителей, Камеры, Титрователя и Макрокоманды).

```
//Считывание события и подсчет вошедших посетителей
if (Event.SourceType == "PEOPLE_COUNTER" && Event.SourceId == "N" && Event.Action ==
"IN") //N - Номер Детектора подсчета посетителей
  {
    i = Itv_var("counter_i");
    k = Itv_var("counter_k");
    i++;
    Itv_var("counter_i")=i;
//Отображение количества посетителей через Титрователь в Мониторе

    DoReactStr("CAM","M","CLEAR_SUBTITLES","title_id<L>"); //M - Номер Камеры  L -
Номер Титрователя
    DoReactStr("CAM","M","ADD_SUBTITLES","command<Кол-во посетителей (вх./вых.): "+i+"
/ "+k+"\r>,page<BEGIN>,title_id<L>"); //M, L - то же

  }
//Считывание события и подсчет вышедших посетителей
if (Event.SourceType == "PEOPLE_COUNTER" && Event.SourceId == "N" && Event.Action ==
"OUT") //N - Номер Детектора подсчета посетителей
  {
    i = Itv_var("counter_i");
    k = Itv_var("counter_k");
    k++;
    Itv_var("counter_k")=k;
//Отображение количества посетителей через Титрователь в Мониторе

   DoReactStr("CAM","M","CLEAR_SUBTITLES","title_id<L>"); //M - Номер Камеры  L -
Номер Титрователя
   DoReactStr("CAM","M","ADD_SUBTITLES","command<Кол-во посетителей (вх./вых.): "+i+" /
"+k+"\r>,page<BEGIN>,title_id<L>"); //M, L - то же
  }
//Обнуление счетчика по Макрокоманде (предварительно в Интеллекте должна быть создана
Макрокоманда)
if (Event.SourceType == "MACRO" && Event.SourceId == "P" && Event.Action == "RUN") //
P - Номер Макрокоманды
  {
```

```
    Itv_var("counter_i")=0;
    Itv_var("counter_k")=0;
    i=0;
    k=0;
//Отображение количества посетителей через Титрователь в Мониторе

    DoReactStr("CAM","M","CLEAR_SUBTITLES","title_id<L>"); //M - Номер Камеры  L -
Номер Титрователя
    DoReactStr("CAM","M","ADD_SUBTITLES","command<Кол-во посетителей (вх./вых.): "+i+"
/ "+k+"\r>,page<BEGIN>,title_id<L>"); //M, L - то же
  }
```

В результате при отображении соответствующей камеры в Мониторе на видеоизображение будет накладываться текстовое сообщение о количестве вошедших и вышедших посетителей.

> **Примечание**
>
> При использовании параметров page<BEGIN> и page<END> будут заполняться соответствующие поля в базе титров, что даст возможность искать данные с помощью интерфейсного объекта **Поиск по титрам**.

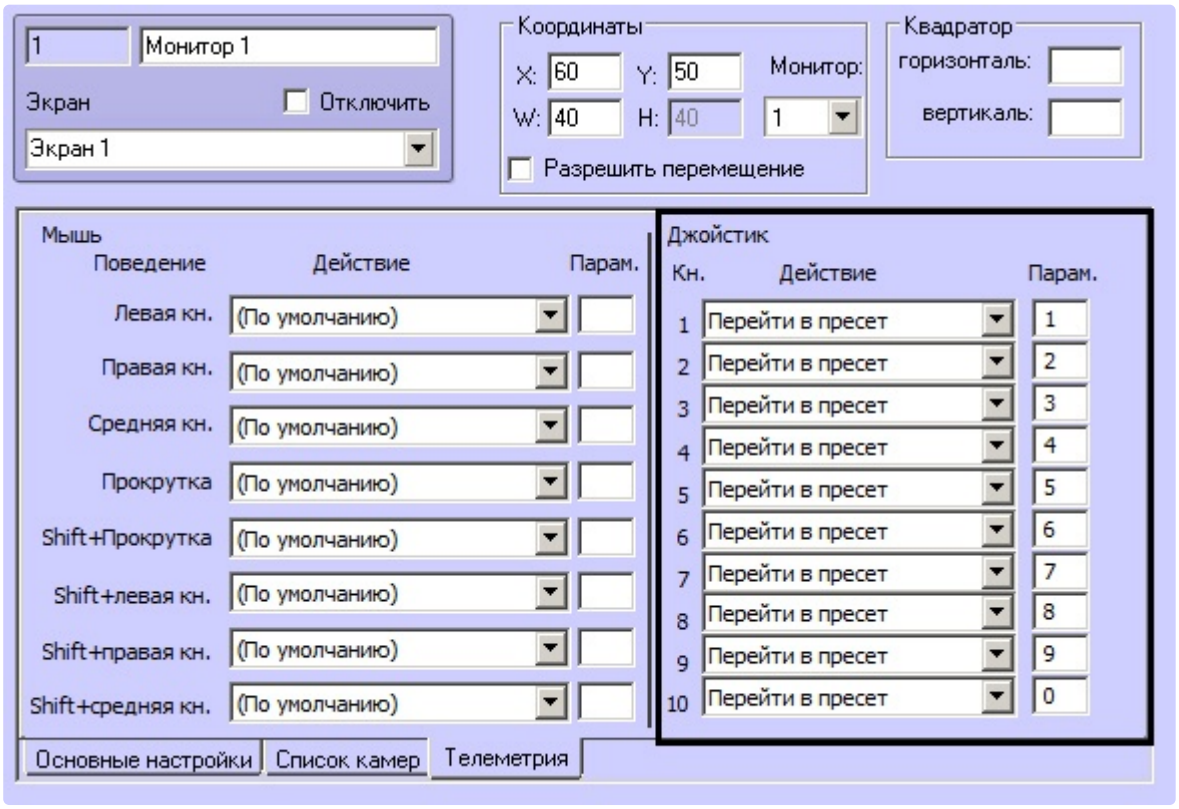
Настройки шрифта, цвета и положения надписи настраиваются на панели настройки объекта **Титрователь** (см. Настройка вывода титров поверх видеоизображения).

Для обнуления счетчика посетителей предварительно на вкладе **Программирование** создается объект **Макрокоманда**, название которой можно для удобства изменить, например, на «Обнуление счетчика посетителей».

Макрокоманду обнуления можно запускать как вручную через главное меню ПК *Интеллект*, так и автоматически в любое заданное время (для этого используется таблица **События** на панели настройки объекта **Макрокоманда**, где необходимо указать предварительно настроенный объект **Временная зона**). Подробные сведения об использовании объектов **Макрокоманда** и **Временная зона** изложены в документе Руководство Администратора.

9.14.1.3 Пример 3. Отображение камеры на мониторе по нажатию кнопки на пульте управления

Приведенный ниже пример работает только для камер, у которых в конфигурации создан пульт управления PTZ. При настройке **Монитора видеонаблюдения** следует для 10 кнопок джойстика выбрать действие **Перейти в пресет** с параметрами 1,2,3...,0 (см. Руководство по установке и настройке компонентов охранной системы, раздел Присваивание клавишам джойстика команд при помощи Монитора видеонаблюдения).



Пример. По нажатию на кнопку пульта отображать соответствующую камеру в активном Мониторе. Скрипт должен срабатывать по таймеру с ID = 1.

> **Примечание**
>
> Необходимо заранее создать и настроить объект **Таймер**, установив значение Год равным текущему году. Настройка объекта **Таймер** подробно описана в документе Руководство администратора, в разделе Создание и настройка объекта Таймер.

После каждого нажатия кнопки на пульте управления необходимо ожидать нажатия следующей кнопки в течение 2 секунд. Если нажатия не произошло, то необходимо выводить на экран камеру с набранным номером.

```
if (Event.SourceType=="TIMER" && Event.SourceId=="1" && Event.Action=="TRIGGER")
{
  mon="1";
  DebugLogString("на монитор "+ Itv_var("cam"));
  DoReactStr("MONITOR",mon,"ACTIVATE_CAM","cam<"+Itv_var("cam")+">");
  Itv_var("cam")="";
}
```

```
if (Event.GetParam("source_type")=="TELEMETRY" && Event.GetParam("action")=="GO_PRESE
T")
{
  DoReactStr("TIMER","1","START","bound<2>");
  var key=Event.GetParam("param4_val");
  DebugLogString("Key:"+key);
  Itv_var("cam")=Itv_var("cam")+key;
  DebugLogString(Itv_var("cam"));
}
```

9.14.1.4 Пример 4. Наложение титров

По макрокоманде 1 выводить текст

«NNN

Titles»

(с переносом строки) поверх видеоизображения камеры 1, используя титрователь 1. По макрокоманде 2 отключать вывод этого текста.

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
    DoReactStr("MONITOR","1","SET_TITLES","titles<NNN \r Titles>,cam<1>,title_id<1>")
;
}

if (Event.SourceType == "MACRO" && Event.SourceId == "2" && Event.Action == "RUN")
{
DoReactStr("MONITOR","1","CLEAR_TITLES","cam<1>,title_id<1>");
}
```

9.14.1.5 Пример 5. Отображение видеокамеры с потоком выбранного типа в окне Монитора видеонаблюдения

По макрокоманде отображать видеокамеру с потоком выбранного типа или с требуемым id потока на Мониторе 2.

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN") //
по типу потока
{
    DoReactStr("MONITOR","2","REMOVE","cam<5>");
    DoReactStr("MONITOR","2","ADD_SHOW","cam<5>,stream_id<stream_client_flag>");
}

if (Event.SourceType == "MACRO" && Event.SourceId == "2" && Event.Action == "RUN") //
по типу потока
{
    DoReactStr("MONITOR","2","REMOVE","cam<5>");
```

```
    DoReactStr("MONITOR","2","ADD_SHOW","cam<5>,stream_id<stream_analytics_flag>");
}

if (Event.SourceType == "MACRO" && Event.SourceId == "3" && Event.Action == "RUN") //
по типу потока
{
    DoReactStr("MONITOR","2","REMOVE","cam<5>");
    DoReactStr("MONITOR","2","ADD_SHOW","cam<5>,stream_id<stream_archive_flag>");
}

if (Event.SourceType == "MACRO" && Event.SourceId == "4" && Event.Action == "RUN") //
по типу потока
{
    DoReactStr("MONITOR","2","REMOVE","cam<5>");
    DoReactStr("MONITOR","2","ADD_SHOW","cam<5>,stream_id<stream_alarm_flag>");
}

if (Event.SourceType == "MACRO" && Event.SourceId == "5" && Event.Action == "RUN") //
по id потока
{
    DoReactStr("MONITOR","2","REMOVE","cam<5>");
    DoReactStr("MONITOR","2","ADD_SHOW","cam<5>,stream_id<5.1>");
}
```

9.14.2 Примеры скриптов с Картой

MAP Карта

9.14.2.1 Задание текста для отображения на карте

При добавлении объекта на карту имеется возможность выбрать способ отображения **Текст** (см. Прикрепление объектов к слою интерактивной карты). При этом для изменения отображаемого текста может использоваться скрипт на языке JScript.

Пример. Для объекта Камера 1 выбран способ отображения на карте Текст. Требуется при выполнении Макрокоманды 1 выводить на карте и в отладочном окне значение переменной MyVar, считанной из файла C:\test.ini.

```
if(Event.SourceType == "MACRO" && Event.Action == "RUN")
{
 var result = parseInt(ReadIni("MyVar","C:\\test.ini"));
 result += 2;
 NotifyEventStr("CAM","1","ANALOG_PARAMS","text<Значение переменной = \n" + result +
">, blink_state<1>");
 DebugLogString(result);
```

```
}
```

9.14.3 Примеры скриптов с Детекторами

CAM_VMDA_DETECTOR Детектор VMDA

CAM_IP_DETECTOR Детектор встроенный

9.14.3.1 Пример 1. Скрипт для выделения оставленных предметов рамкой на живом видео

Если используется функция детекции предметов на видеоизображении (см. Создание и настройка объекта Трекер), то при просмотре архива обнаруженные оставленные предметы будут выделяться на видеоизображении рамкой. Для выделения оставленных предметов рамкой на живом видео необходимо использовать следующий скрипт, который предназначен для рисования рамки вокруг оставленного предмета при получении тревоги по детектору VMDA:

```
if (Event.SourceType=="CAM_VMDA_DETECTOR")
{
 cam=GetObjectParentId("CAM_VMDA_DETECTOR",Event.SourceId,"CAM");
 if (Event.Action=="ALARM")
 {
   var x1,x2,y1,y2;
   x1=Event.GetParam("x");
   x2=Event.GetParam("w");
   y1=Event.GetParam("y");
   y2=Event.GetParam("h");
   x2=parseInt(x1)+parseInt(x2);
   y2=parseInt(y1)+parseInt(y2);
   DoReactStr("MONITOR","","SET_MARKRECT","cam<"+cam+">,color<255>,id<"+cam+">,x1<"+x
1+">,x2<"+x2+">,y1<"+y1+">,y2<"+y2+">");
   DebugLogString("x1:"+x1+" x2:"+x2+" y1:"+y1+" y2:"+y2);
 }
 else
 {
   DoReactStr("MONITOR","","DEL_MARKRECT","cam<"+cam+">,id<"+cam+">");
 }
}
```

9.14.3.2 Пример 2. Использование встроенного детектора подсчета посетителей IP-камеры Bosch FLEXIDOME IP dynamic 7000 VR

По достижению количества посетителей 20 на встроенном детекторе подсчета посетителей IP-камеры Bosch FLEXIDOME IP dynamic 7000 VR (с идентификатором 1) вызывать макрокоманду 1.

```
n=20;
if(Event.SourceType == "CAM_IP_DETECTOR" && Event.SourceId=="1" && Event.Action ==
"DETECTED")
{
 v=Event.GetParam("param0").split(";")[1];
 if (parseInt(v.split(":")[1])==n)
 {
     DoReactStr("MACRO","1","RUN","");
 }
}
```

9.14.4 Примеры скриптов с Макрокомандами

MACRO Макрокоманда

9.14.4.1 Пример 1. Отправка камере команды через HTTP API камеры

IP-адрес камеры 192.168.0.13.

Следующая команда позволяет включить стеклоочиститель на камере:

`192.168.10.101/httpapi/SendPTZ?action=sendptz&PTZ_PRESETSET=85`

Следующая команда выключает стеклоочиститель на камере:

`192.168.10.101/httpapi/SendPTZ?action=sendptz&PTZ_PRESETSET=86`

Необходимо отправлять данные команды камере при помощи скрипта на языке JScript.

```
function DoPreset(preset)
{
    xmlhttp=new ActiveXObject("MSXML2.XMLHTTP");
    if(xmlhttp == null)
    {
           return;
    }
```

```
    xmlhttp.open("GET", "http://192.168.0.13/httpapi/SendPTZ?
action=sendptz&PTZ_PRESETSET="+preset, false,"admin","1234");

    xmlhttp.send();
    DebugLogString(xmlhttp.status);
}
if (Event.SourceType == "MACRO" && Event.SourceId == "6" && Event.Action == "RUN")
{
 DoPreset("85");
}

if (Event.SourceType == "MACRO" && Event.SourceId == "7" && Event.Action == "RUN")
{
 DoPreset("86");
}
```

9.14.4.2 Пример 2. Отправка сообщения электронной почты с HTML-разметкой

По макрокоманде 1 отправлять сообщение с приложенными файлами detected.png и found.jpg из папки C:\\Pictures\\ на адрес example@gmail.com. Сообщение должно быть отформатировано следующим образом:

Обнаружено лицо

Обнаруженное лицо	Лицо в базе данных
файл detected.png	файл found.jpg

```
if(Event.SourceType == "MACRO" && Event.SourceId=="1" &&Event.Action=="RUN")
{
var file1 = "detected.png";
var file2 = "found.jpg";
var file_folder = "C:\\Pictures\\";

var test_event = CreateMsg();
test_event.StringToMsg("MAIL_MESSAGE|1|SEND_RAW|
cc<>,to<example@gmail.com>,objname<Почтовое сообщение
 1>,subject<>,parent_id<1>,flags<>,pack<>,name<Почтовое сообщение
 1>,from<server@itv.ru>,_marker<>");

test_event.SetParam("body","<html><body>\r\n<h3>Face found</
h3>\r\n<table><tr>\r\n<td>Detected</td><td>Found in DB</td>\r\n</tr><tr>\r\n<td><img
width='200' alt='image1' src='cid:"
+
file1
+
"'/></td>\r\n<td><img width='200' alt='image2' src='cid:"
```

```
+
file2
+
"'/></td>\r\n</tr><tr>\r\n<td>Сообщение отправлено из ПК Интеллект</td>\r\n</tr></
table>\r\n</body></html>");
test_event.SetParam("attachments",file_folder+file1+";" + file_folder+file2);
test_event.SetParam("is_body_html", 1);
DoReact(test_event);
}
```

9.14.5 Пример скрипта с Пользователями

PERSON Пользователь

CORE Ядро

MACRO Макрокоманда

9.14.5.1 Создание тестовых пользователей

По макрокоманде 101 создать в ПК *Интеллект* 50 пользователей с идентификаторами от 100 до 150, назначив им уровень доступа с идентификатором 1 (при условии, что уровень доступа назначен отделу, в который добавляются пользователи, и пользователи наследуют уровень доступа отдела) и привязав карту доступа с номером, равным идентификатору пользователя. Номер карты должен быть в HEX-формате. В отделе должно быть не более 30 пользователей (для ускорения процесса добавления).

Примечание.

Дополнительную информацию об уровнях доступа и картах доступа см. в документации на ПК *ACFA-Интеллект* в хранилище документации Документация для продуктов компании ITV.

Если в ПК *Интеллект* настроена интеграция СКУД, поддерживающая динамическую запись пользователей, то при отправке события CORE||UPDATE_OBJECT|objtype<PERSON> создаваемый пользователь будет автоматически записываться в контроллер СКУД. Если динамика не поддерживается, то запись пользователей в контроллер необходимо будет инициировать вручную.

```
dep=10; // идентификатор отдела
start=100;// идентификатор первого пользователя
last=150; // идентификатор последнего пользователя
```

```
acc_lev=1; // идентификатор уровня доступа
dep_count=30; // максимальное количество пользователей в отделе

if( Event.SourceType == "MACRO" && Event.Action == "RUN" && Event.SourceId=="101")
 {
    kol=0;
    card_count=0;
    NotifyEventStr("CORE","","UPDATE_OBJECT","objtype<DEPARTMENT>,objid<"+dep+">");
    for (i=start;i<=last;i++)
    {
        kol++;
        card_count++;
        card=decToHex(card_count);
        if (card[card.length-1]==0)
        {
            card_count++;
            card=decToHex(card_count);
        }

        if (kol==dep_count)
        {
            NotifyEventStr("CORE","","UPDATE_OBJECT", "objtype<PERSON>,objid<"+i+">,n
ame<user"+i+">,parent_id<"+dep+">, level_id<"+acc_lev+">, facility_code<0>, card<"+ca
rd+">");
            kol=0;
            dep++;
            NotifyEventStr("CORE","","UPDATE_OBJECT","objtype<DEPARTMENT>,objid<"+dep
+">");
        }
        else
        {
            NotifyEventStr("CORE","","UPDATE_OBJECT", "objtype<PERSON>,objid<"+i+">,n
ame<user"+i+">,parent_id<"+dep+">,level_id<"+acc_lev+">, facility_code<0>,card<"+card
+">");
        }
        Sleep(10);
    }
 }

function decToHex(n)
{
 return Number(n).toString(16);
}
```

9.14.6 Примеры скриптов с Сервером и Менеджером инцидентов

INC_SERVER Сервер инцидентов

INC_MANAGER Менеджер инцидентов

9.14.6.1 Пример 1. По макрокоманде 1 изменить статус события Тревога камеры 1 на Завершено.

```
OnEvent("MACRO","1","RUN")
{
    DoReactStr("INC_SERVER","1","UPDATE_STATUS","status<3>,objtypes<CAM>,objids<1>,ac
tions<MD_START>");
}
```

Пример 2. По макрокоманде 2 на Сервере инцидентов 1 изменить статус эскалации событий камеры 1 на Ожидает обработки (не эскалировано).

```
if (Event.SourceType == "MACRO" && Event.SourceId == 2 && Event.Action == "RUN")
{
DoReactStr("INC_SERVER","1","UPDATE_ESCALATE_STATUS","escalated<0>,objtypes<CAM>,objid
ds<1>");
}
```

9.14.6.2 Пример 2. Изменение статуса события в Менеджере инцидентов

При работе с объектами в **Менеджере инцидентов** имеется возможность изменять статус события объекта (см. Обработка событий). При этом для изменения статуса событий объектов может использоваться скрипт на языке JScript.

Пример. По макрокоманде 3 изменить статус события Тревога камеры 1 или 2 на Завершено.

```
if (Event.SourceType == "MACRO" && Event.SourceId == 3 && Event.Action == "RUN")
 {
    DoReactStr("INC_SERVER","1","UPDATE_STATUS","status<3>,objtypes<CAM>,objids<1|
2>,actions<MD_START|MD_START>");
 };
```

9.14.7 Пример скрипта с Сервисом отказоустойчивости

FAILOVER Сервис отказоустойчивости

9.14.7.1 Использование событий START и STOP для Сервиса отказоустойчивости

Требуется не переносить на резервный Сервер объекты с более чем одного основного Сервера. Для этого при переносе на резервный Сервер объектов с какого-либо основного Сервера необходимо отключать все остальные объекты **Сервис отказоустойчивости** на данном резервном Сервере.

```
if (Event.SourceType == "FAILOVER")
{
if (Event.Action == "START") {action="DISABLE";}
if (Event.Action == "STOP") {action="ENABLE";}
id=Event.SourceId;
msg=CreateMsg();
  msg.StringToMsg(GetObjectIds("FAILOVER"));
  var objCount=msg.GetParam("id.count");
    for (i=0;i<objCount;i++)
    {
     pid=msg.GetParam("id."+i);
     if (!(id==pid)) { DoReactStr("FAILOVER",pid,action,"");}
  }
}
```

9.14.8 Примеры скриптов с BacNet

BACNET BacNet

9.14.8.1 Пример 1. Запись в объект с помощью скрипта

```
var msg = CreateMsg();

//bacnet_application_tag
var BACNET_APPLICATION_TAG_NULL = 0;
var BACNET_APPLICATION_TAG_BOOLEAN = 1;
var BACNET_APPLICATION_TAG_UNSIGNED_INT = 2;
var BACNET_APPLICATION_TAG_SIGNED_INT = 3;
var BACNET_APPLICATION_TAG_REAL = 4;
var BACNET_APPLICATION_TAG_DOUBLE = 5;
var BACNET_APPLICATION_TAG_OCTET_STRING = 6;
var BACNET_APPLICATION_TAG_CHARACTER_STRING = 7;
var BACNET_APPLICATION_TAG_BIT_STRING = 8;
```

```
//bacnet_objtype
var OBJECT_ANALOG_INPUT = 0;
var OBJECT_ANALOG_OUTPUT = 1;
var OBJECT_ANALOG_VALUE = 2;
var OBJECT_BINARY_INPUT = 3;
var OBJECT_BINARY_OUTPUT = 4;
var OBJECT_BINARY_VALUE = 5;

//bacnet_property_id
var PROP_PRESENT_VALUE = 85;

msg.StringToMsg("BACNETINT|1|WRITE");
msg.SetParam("bacnet_application_tag", BACNET_APPLICATION_TAG_UNSIGNED_INT);
msg.SetParam("bacnet_value",30);

msg.SetParam("bacnet_objtype",OBJECT_ANALOG_VALUE);
msg.SetParam("bacnet_instance",0);

msg.SetParam("bacnet_property_id",PROP_PRESENT_VALUE);
msg.SetParam("bacnet_device_id",12345);

DoReact(msg);
```

В случае успешного выполнения скрипта в **Отладочном окне** появится событие:

```
Event:
BACNETINT|1|WRITE_OCCURES|
sender<Udp:47808>,slave_id<ASUS>,fraction<186>,invoke_id<43>,owner<ASUS>,module<
bacnetint.vshost.exe>,date<27-11-18>,
value<PROP_PRESENT_VALUE>,guid_pk<{E23BD6CB-19F2-E811-8B83-C860008A29F9}
>,object_id<OBJECT_ANALOG_VALUE:0>,
core_global<1>,adr<192.168.0.197:56747>,time<10:55:33>,source_guid<557367ce-19f2
-e811-8b83-c860008a29f9>
```

9.14.8.2 Пример 2. Генерация события

```
DebugLogString("Script2");
var msg = CreateMsg();


msg.StringToMsg("BACNETINT|1|EVENT");

msg.SetParam("event_type", "0");
msg.SetParam("from_state", "1");
msg.SetParam("to_state", "0");

msg.SetParam("message_text", "test_text1!");
```

```
DoReact(msg);
```

Если модуль получит событие, то в **Отладочном окне** будет отображено следующее событие:

```
Event:
BACNETINT|1|EVENT_OCCURES|
sender<Udp:47808>,slave_id<ASUS>,fraction<683>,owner<ASUS>,event_type<EVENT_CHAN
GE_OF_BITSTRING>,module<bacnetint.vshost.exe>,
message_text<test_text1!>,date<27-11-18>,guid_pk<{6D34BA08-1CF2-E811-8B83-
C860008A29F9}>,from_state<EVENT_STATE_FAULT>,
core_global<1>,adr<192.168.0.197:57878>,to_state<EVENT_STATE_NORMAL>,time<11:11:
34>,source_guid<bd51a40d-1cf2-e811-8b83-c860008a29f9>
```

9.14.8.3 **Пример 3. Считывание показателей с объекта**

```
var msg = CreateMsg();

//bacnet_application_tag
var BACNET_APPLICATION_TAG_NULL = 0;
var BACNET_APPLICATION_TAG_BOOLEAN = 1;
var BACNET_APPLICATION_TAG_UNSIGNED_INT = 2;
var BACNET_APPLICATION_TAG_SIGNED_INT = 3;
var BACNET_APPLICATION_TAG_REAL = 4;
var BACNET_APPLICATION_TAG_DOUBLE = 5;
var BACNET_APPLICATION_TAG_OCTET_STRING = 6;
var BACNET_APPLICATION_TAG_CHARACTER_STRING = 7;
var BACNET_APPLICATION_TAG_BIT_STRING = 8;

//bacnet_objtype
var OBJECT_ANALOG_INPUT = 0;
var OBJECT_ANALOG_OUTPUT = 1;
var OBJECT_ANALOG_VALUE = 2;
var OBJECT_BINARY_INPUT = 3;
var OBJECT_BINARY_OUTPUT = 4;
var OBJECT_BINARY_VALUE = 5;
var OBJECT_CHARACTERSTRING_VALUE = 40;

//bacnet_property_id
var PROP_PRESENT_VALUE = 85;

msg.StringToMsg("BACNETINT|1|READ");

msg.SetParam("bacnet_objtype",OBJECT_ANALOG_INPUT);
msg.SetParam("bacnet_instance",0);
msg.SetParam("bacnet_property_id",PROP_PRESENT_VALUE);
msg.SetParam("bacnet_device_id",123456);
DoReact(msg);
```

При успешном считывании в **Отладочном окне** будет отражено событие:

```
Event:
BACNETINT|1|READ_RESULT|
slave_id<example>,fraction<387>,owner<example>,module<bacnetint.run>,date<10-11-
21>,
guid_pk<{622928D6-3349-EC11-96F2-309C23D50163}
>,core_global<1>,bacnet_value<20,8>,
time<15:26:03>,param0<ok>,source_guid<e8f7ded1-3349-ec11-96f2-309c23d50163>
```

9.14.9 Пример с ботом Telegram

TELEGRAM Telegram бот

9.14.9.1 Отправка сообщения в Telegram

По макрокоманде 1 отправлять с помощью бота Telegram текст, изображение или геолокацию.

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")

{

  //Sending with chat_id & bot_id from object settings:

  DoReactStr("TELEGRAM","1","SEND","text<Intellect works great>");

  //Explicit setting chat_id & bot_id in the command:

  DoReactStr("TELEGRAM","1","SEND","text<Intellect works
great>,chat_id<828752651>,bot_id<809045046:AAGtKxtDWu5teRGKW_Li8wFBQuJ-l4A9h38>");

  //Sending file with chat ID and bot ID:

  DoReactStr("TELEGRAM",1,"SENDPHOTO","caption<Intellect works
great>,chat_id<828752651>,bot_id<809045046:AAGtKxtDWu5teRGKW_Li8wFBQuJ-
l4A9h38>,photo<G:\\1.jpg>");

  //Sending geolocation with chat ID and bot ID:

  DoReactStr("TELEGRAM",1,"SEND","text<Hello
world>,chat_id<828752651>,bot_id<809045046:AAGtKxtDWu5teRGKW_Li8wFBQuJ-
l4A9h38>",longtitude<37.3428359>,latitude<55.6841654>,address<Office>);

}
```

9.14.10 Примеры скриптов с Протоколом событий

EVENT_VIEWER Протокол событий

9.14.10.1 Активация одного фильтра

Пример. Активировать в **Протоколе событий** фильтр Filter 1 для Протокола событий 1 с помощью Макрокоманды 1.

```
if(Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
DoReactStr("EVENT_VIEWER","1","SET_FILTERS","filter0<Filter 1>");
}
```

9.14.10.2 Активация нескольких фильтров

Пример. Активировать в **Протоколе событий** фильтры Filter 1, Filter 3, Filter 5 для Протокола событий 1 с помощью Макрокоманды 4.

```
if(Event.SourceType == "MACRO" && Event.SourceId == "4" && Event.Action == "RUN")
{
DoReactStr("EVENT_VIEWER","1","SET_FILTERS","filter0<Filter 1>,filter1<Filter
3>,filter2<Filter 5>");
}
```

9.15 Приложение 1. Описание утилиты Редактор-Отладчик

9.15.1 Назначение утилиты Редактор-Отладчик

Утилита *Редактор-Отладчик* предназначена для создания, редактирования и отладки скриптов в программном комплексе *Интеллект*.

Утилита *Редактор-Отладчик* обеспечивает выполнение следующих задач:

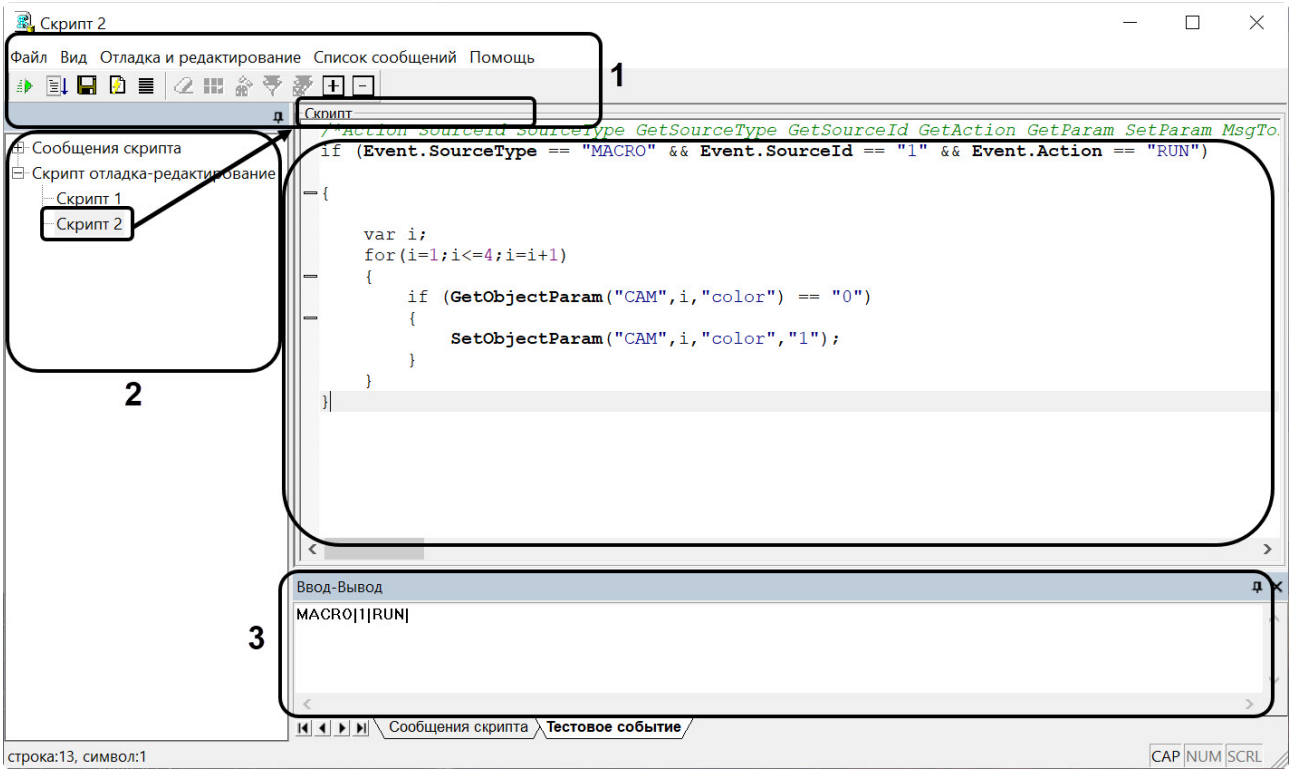
1. Создание и редактирование скриптов с использованием встроенного текстового редактора.
2. Отладка скрипта посредством встроенного отладочного окна.
3. Использование фильтра вывода информации в отладочном окне.

4. Создание и использование тестового события для отладки скрипта.
5. Сохранение скрипта на жесткий диск.
6. Загрузка скрипта с жесткого диска.

9.15.2 Описание интерфейса утилиты Редактор-Отладчик

9.15.2.1 Интерфейс утилиты Редактор-Отладчик

Пользовательский интерфейс утилиты *Редактор-отладчик* представлен главным меню и панелью инструментов (1), списком объектов (2) и панелью редактирования/просмотра (3).



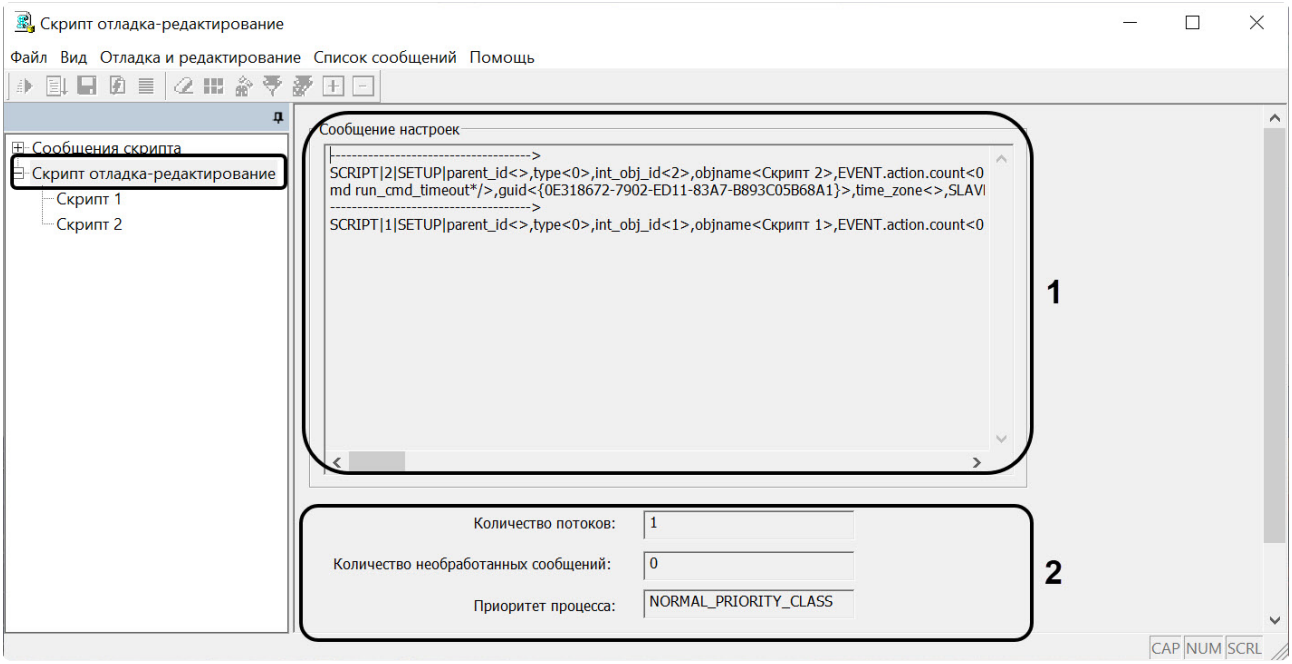
9.15.2.2 Вкладка Скрипт отладка-редактирование

На странице:

9.15.2.2.1 Описание интерфейса вкладки Скрипт отладка-редактирование

Вкладка **Скрипт отладка-редактирование** предназначена для редактирования скриптов и создания тестовых событий.

Интерфейс вкладки **Скрипт отладка-редактирование**:



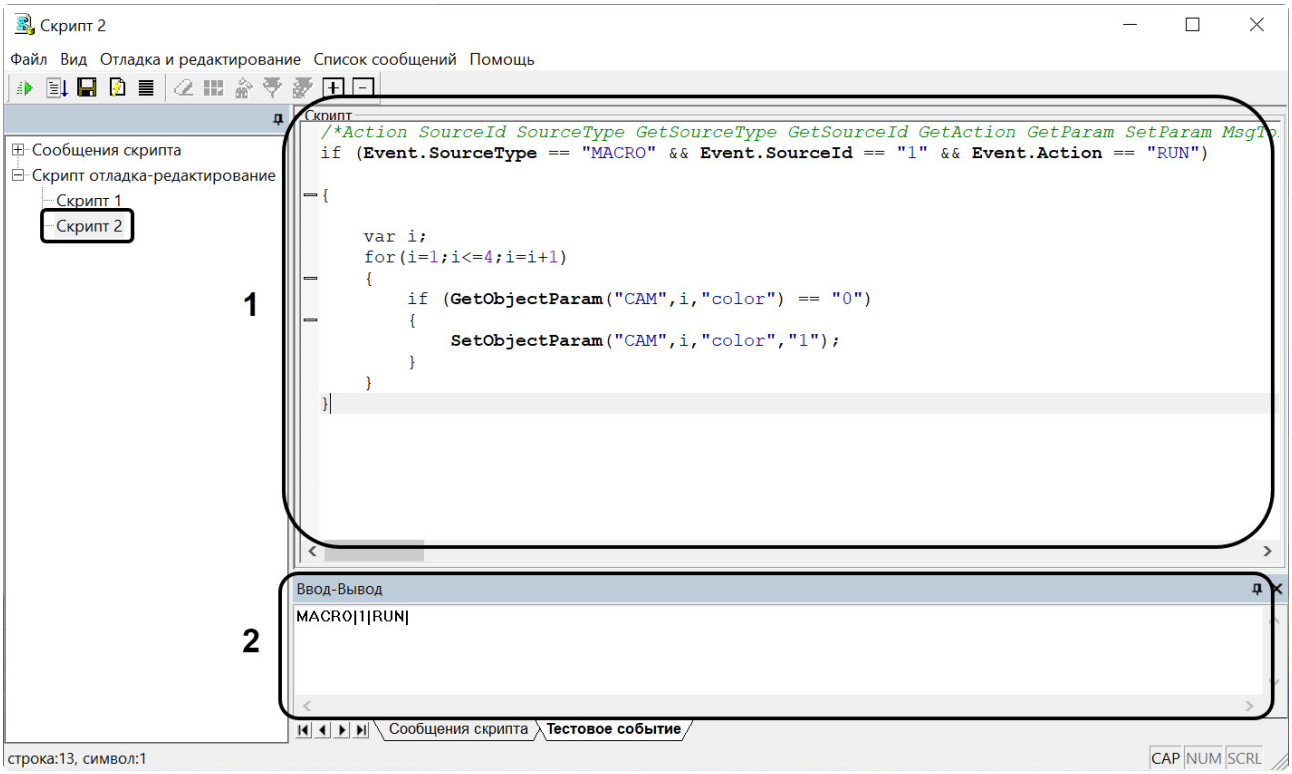
Описание интерфейса вкладки **Скрипт отладка-редактирование**:

Цифра на изображении	Название параметра	Способ задания значения параметра	Описание параметра
1	Сообщение настроек	Автоматически	Информация об инициализации объектов **Скрипт** в системе
2	Дополнительная информация	Автоматически	Дополнительная информация о скриптах

9.15.2.2.2 Описание интерфейса объекта Скрипт (вкладка Скрипт отладка-редактирование)

Объект **Скрипт** вкладки **Скрипт отладка-редактирование** предназначен для создания и редактирования скриптов и тестового события.

Интерфейс объекта **Скрипт**:



Описание интерфейса объекта **Скрипт**:

Цифра на изображении	Название параметра	Способ задания значения параметра	Описание параметра	Используемые символы	Значение по умолчанию	Диапазон значений
1	Скрипт	Ввод значения в поле	Содержит текст скрипта	Латинский алфавит, кириллица и служебные символы	Пустая строка	Неограниченное количество символов

Цифра на изображении	Название параметра	Способ задания значения параметра	Описание параметра	Используемые символы	Значение по умолчанию	Диапазон значений
2	Ввод-Вывод	Ввод значения в поле	Содержит текст тестового события	Латинский алфавит, кириллица и служебные символы	Пустая строка	Неограниченное количество символов

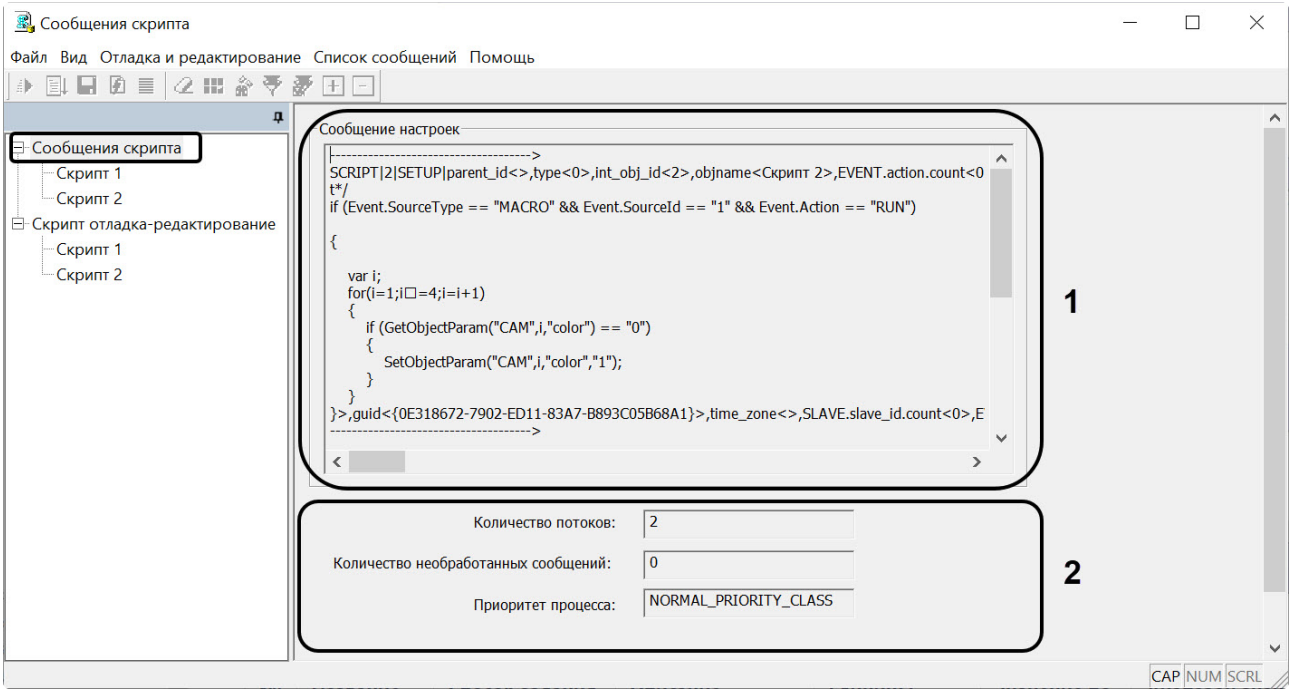
9.15.2.3 Вкладка Сообщения скрипта

На странице:

9.15.2.3.1 Описание интерфейса вкладки Сообщения скрипта

Вкладка **Сообщения скрипта** предназначена для отображения отладочных окон скриптов.

Интерфейс вкладки **Сообщения скрипта**:



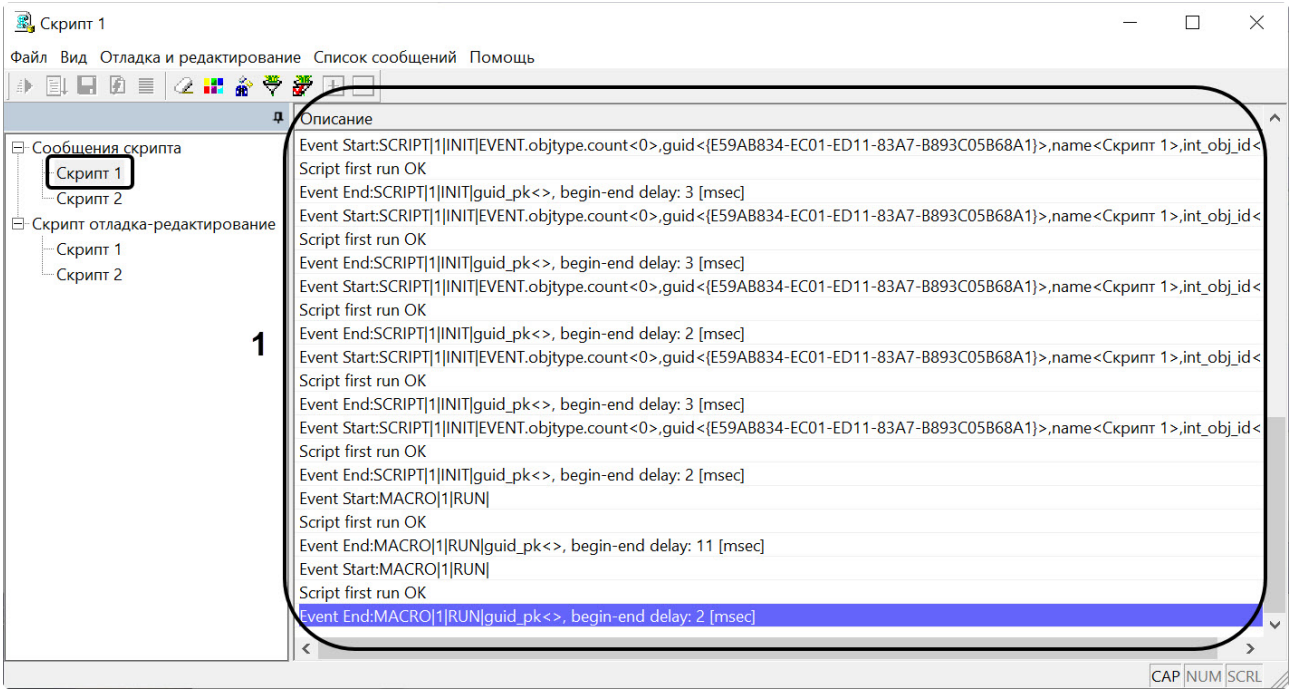
Описание интерфейса вкладки **Сообщения скрипта**:

Цифра на изображении	**Название параметра**	**Способ задания значения параметра**	**Описание параметра**
1	Сообщение настроек	Автоматически	Информация об инициализации объектов **Скрипт** в системе
2	Дополнительная информация	Автоматически	Дополнительная информация о скриптах

9.15.2.3.2 Описание интерфейса объекта Скрипт (вкладка Сообщения скрипта)

Объект **Скрипт** вкладки **Сообщения скрипта** предназначен для отображения системных, тестовых и пользовательских событий, относящихся к созданным в ПК *Интеллект* скриптам.

Интерфейс объекта **Скрипт**:



Описание интерфейса объекта **Скрипт** представлено в таблице.

Цифра на изображении	Название параметра	Способ задания значения параметра	Описание параметра	Единицы измерения	Значение по умолчанию	Диапазон значений
1	Описание	Автоматически	Отображает информацию о событиях, происходящих в системе	Латинский алфавит, кириллица и служебные символы	На задано	По умолчанию в данном списке отображается 200 строк. Изменить это значение можно при помощи ключа реестра DebugMaxLines (см. Справочник ключей реестра)

9.15.2.4 Главное меню

9.15.2.4.1 Описание интерфейса главного меню

Главное меню утилиты *Редактор-Отладчик* предназначено для вызова команд, реализуемых утилитой. Команды разделены по функциональным признакам на группы и размещены в следующих пунктах меню: **Файл, Вид**, **Отладка и редактирование**, **Список сообщений**, **Помощь.**

Описание пунктов главного меню утилиты:

№ п/п	Название параметра	Способ задания значения параметра	Описание параметра
1	Файл	Выбор команды из раскрывающегося списка	Содержит список команд, обеспечивающих сохранение и загрузку скриптов, завершение работы с утилитой
2	Вид		Содержит список команд, обеспечивающих отображение в окне утилиты панели инструментов и строки состояния
3	Отладка и редактирование		Содержит список команд, обеспечивающих процесс отладки скрипта
4	Список сообщений		Содержит список команд, позволяющих изменять параметры отображения сообщений в отладочных окнах утилиты
5	Помощь		Содержит команду **О программе...**. По данной команде осуществляется вызов окна, содержащего общие сведения об утилите *Редактор-Отладчик*

9.15.2.4.2 Описание пункта главного меню Файл

Пункт главного меню **Файл** предназначен для вызова команд сохранения и загрузки скриптов, завершения работы с утилитой.

Описание элементов пункта главного меню **Файл**:

№ п/п	Название параметра	Описание параметра
1	Сохранить в базе	Осуществляет сохранение скрипта в соответствующий ему системный объект
2	Сохранить на диск	Осуществляет сохранение скрипта в текстовый файл на жестком диске
3	Загрузить с диска	Осуществляет загрузку скрипта из выбранного текстового файла в редактор утилиты
4	Выход	Команда предназначена для завершения работы с утилитой и закрытия ее диалогового окна

9.15.2.4.3 Описание пункта главного меню Вид

Пункт главного меню **Вид** предназначен для вызова команд, включающих и отключающих отображение в окне утилиты панели инструментов и строки состояния.

Описание элементов пункта главного меню **Вид**:

№ п/п	Название параметра	Способ задания значения параметра	Описание параметра	Значение по умолчанию
1	Инструменты	Установка флажка	Включает или отключает отображение панели инструментов в окне утилиты	Да
2	Строка состояния	Установка флажка	Включает или отключает отображение строки состояния в нижней части окна утилиты	Да

9.15.2.4.4 Описание пункта главного меню Отладка и редактирование

Пункт главного меню **Отладка и редактирование** предназначен для вызова команд отладки скриптов.

Описание элементов пункта главного меню **Отладка и редактирование**:

№ п/п	Название параметра	Описание параметра
1	Тестовый пуск	Производит запуск скрипта по тестовому событию
2	Тестовый запуск с отладчиком	Производит запуск скрипта по тестовому событию с использованием сторонней программы-отладчика
3	Редактировать тестовое событие	Вызывает диалоговое окно, предназначенное для редактирования тестового события
4	Сводная информация	Вызывает отладочное окно **Информация от потока**, в котором отображаются системные, тестовые и пользовательские сообщения, относящиеся только к отлаживаемому скрипту
5	Перейти к строке	Вызывает диалоговое окно, предназначенное для задания номеров символа и строки в тексте скрипта, к которым требуется осуществить переход

9.15.2.4.5 Описание элементов пункта главного меню Список сообщений

Пункт главного меню **Список сообщений** предназначен для изменения параметров отображения сообщений в отладочном окне.

Описание элементов пункта меню **Список сообщений**:

№ п/п	Название параметра	Описание параметра
1	Очистить	Очищает поле **Описание** отладочного окна
2	Цвета	Вызывает окно, предназначенное для задания слов (или других последовательностей символов) в строках, их содержащих, которые требуется выделять цветом в поле **Описание** отладочного окна
3	Поиск	Осуществляет поиск слова (или других последовательностей символов) в поле **Описание** отладочного окна

№ п/п	Название параметра	Описание параметра
4	Установить фильтр	Вызывает окно, с помощью которого осуществляется включение и настройка фильтра. Строки, которые содержат (не содержат) указанные слова должны обязательно быть включены в отладочное окно (или исключены из него)
5	Применить фильтр	Активирует созданный фильтр

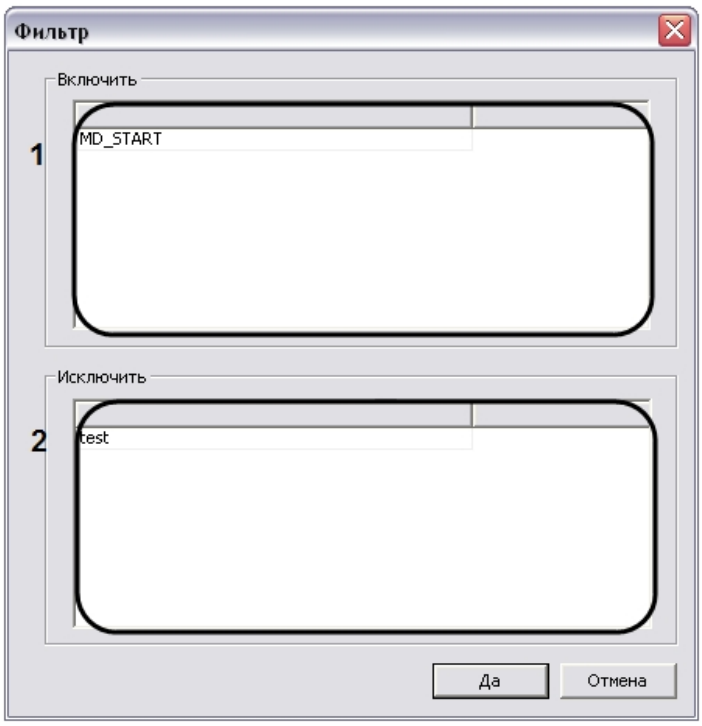
9.15.2.5 Описание диалогового окна Фильтр

Диалоговое окно **Фильтр** предназначено для включения и настройки фильтров сообщений, отображаемых в поле **Описание отладочного окна**.

Интерфейсное окно **Фильтр** вызывается двумя способами:

1. нажатием кнопки **Фильтр** , расположенной на панели инструментов утилиты *Редактор-Отладчик*;
2. по команде **Список сообщений** → **Фильтр**.

Интерфейс окна **Фильтр**:



Описание интерфейса диалогового окна **Фильтр**:

Цифра на изображении	**Название параметра**	**Способ задания значения параметра**	**Описание параметра**	**Используемые символы**	**Значение по умолчанию**	**Диапазон значений**
1	Включить	Ввод значения в поле	Строки, содержащие указанные в данном поле слова (или другие последовательности символов), будут отображаться в отладочном окне	Латинский алфавит, кириллица и служебные символы	Пустая строка	Неограниченное количество символов
2	Исключить	Ввод значения в поле	Строки, содержащие указанные в данном поле слова (или другие последовательности символов), будут исключены из отладочного окна	Латинский алфавит, кириллица и служебные символы	Пустая строка	Неограниченное количество символов

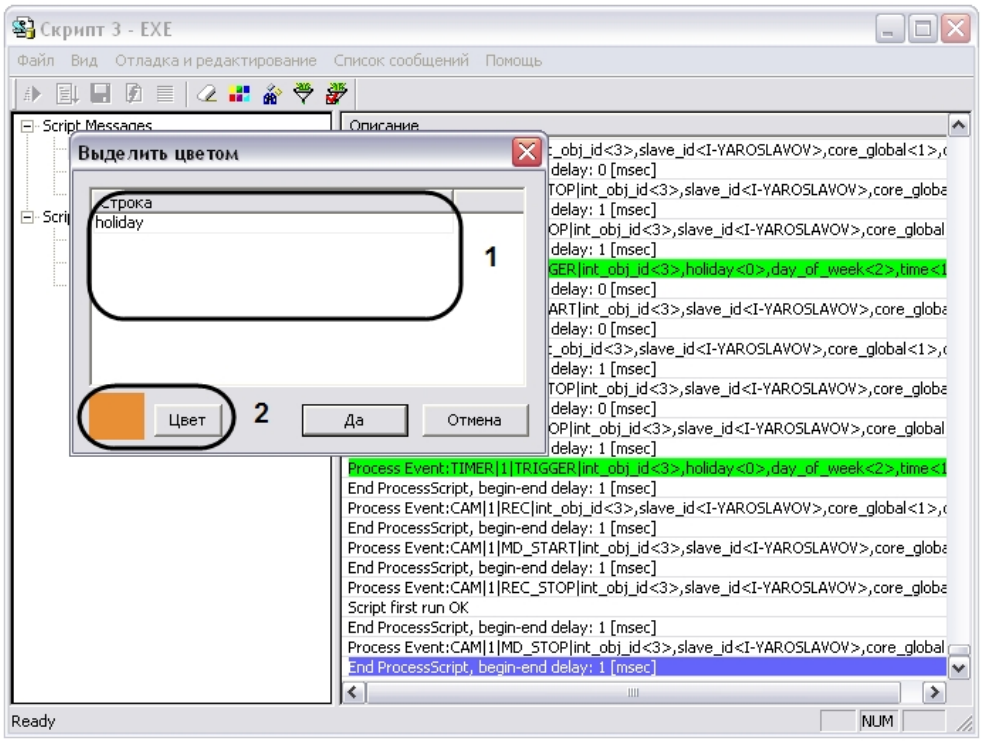
9.15.2.6 Описание диалогового окна Выделить цветом

Диалоговое окно **Выделить цветом** предназначено для настройки выделения в отладочном окне цветом строк, содержащих заданные слова.

Диалоговое окно **Выделить цветом** вызывается двумя способами:

1. нажатием кнопки **Цвета**  , расположенной на панели инструментов утилиты *Редактор-Отладчик*;
2. по команде **Список сообщений → Цвета.**

Интерфейс окна **Выделить цветом**:



Описание элементов диалогового окна **Выделить цветом**:

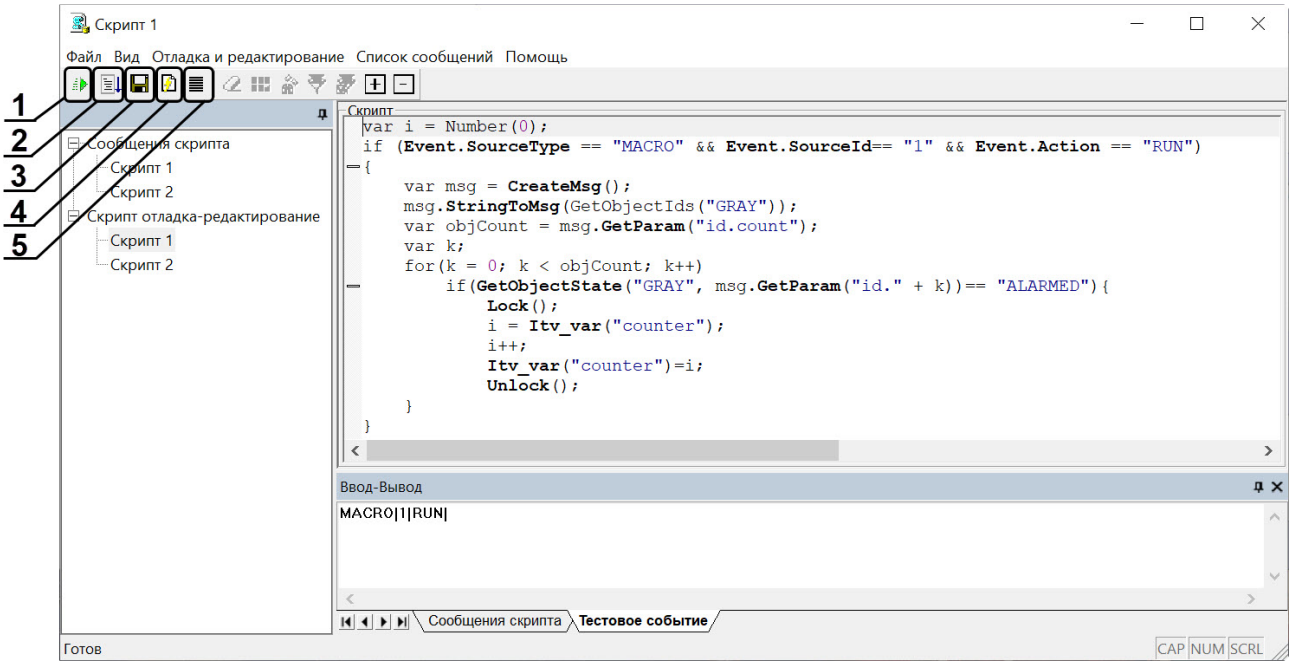
Цифра на изображении	**Название параметра**	**Способ задания значения параметра**	**Описание параметра**	**Единицы измерения**	**Значение по умолчанию**	**Диапазон значений**
1	Строка	Ввод значения в поле	Задает слова (или другие последовательности символов) для выделения содержащих их строк в отладочном окне	Латинский алфавит, кириллица и служебные символы	Пустая строка	Неограниченное количество символов
2	Цвет	Выбор из списка значений	Задает цвет выделения строк в отладочном окне	RGB	Серый	Цветовой диапазон RGB формата

9.15.2.7 Описание панели инструментов утилиты Редактор-Отладчик

Панель инструментов утилиты *Редактор-Отладчик* предназначена для вызова часто используемых функций утилиты.

Панель инструментов утилиты *Редактор-Отладчик* функционирует в двух режимах: с активными кнопками управления скриптами или с активными кнопками управления функциями отладочного окна. Активность кнопок зависит от того, какая вкладка утилиты *Редактор-Отладчик* является активной в данный момент: вкладка **Скрипт отладка-редактирование**, используемая для редактирования скриптов, либо вкладка **Сообщения скрипта**, используемая для просмотра сообщений в отладочном окне.

Интерфейс панели инструментов *Редактор-Отладчик* в режиме редактирования скрипта:

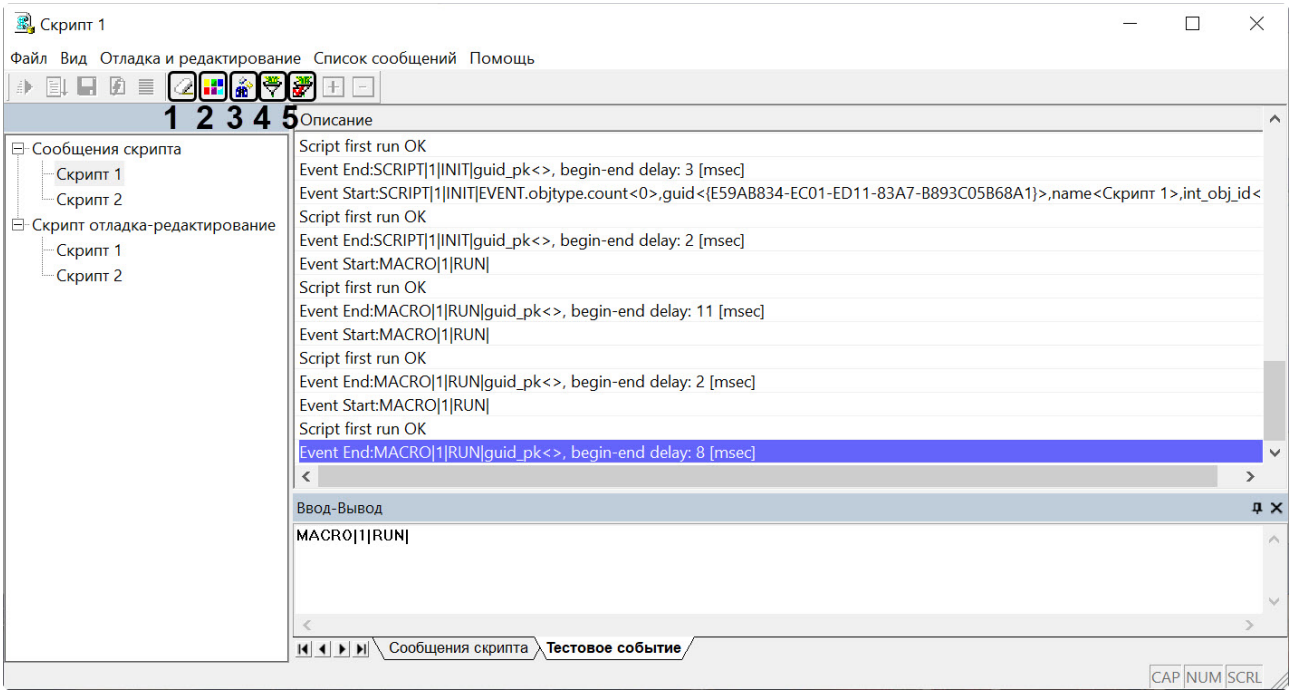


Описание интерфейсов панели инструментов утилиты *Редактор-Отладчик* в режиме редактирования скриптов:

Цифра на изображении	Название параметра	Описание параметра
1	Тестовый запуск	Производит запуск скрипта по тестовому событию
2	Тестовый запуск с отладчиком	Производит запуск скрипта по тестовому событию с использованием сторонней программы-отладчика

Цифра на изображении	Название параметра	Описание параметра
3	Сохранить	Осуществляет сохранение скрипта в системный объект **Скрипт**
4	Редактировать тестовое событие	Вызывает диалоговое окно, предназначенное для редактирования тестового события
5	Сводная информация	Вызывает отладочное окно **Информация от потока**, в котором отображаются системные, тестовые и пользовательские сообщения, относящиеся только к отлаживаемому скрипту

Интерфейс панели инструментов утилиты *Редактор-Отладчик* в режиме работы с отладочным окном:



Описание интерфейса панели инструментов утилиты *Редактор-Отладчик* в режиме работы с отладочным окном:

Цифра на изображении	Название параметра	Описание параметра
1	Очистить	Очищает поле **Описание** отладочного окна
2	Цвета	Вызывает окно, предназначенное для задания слов (или других последовательностей символов), строки, содержащие которые, требуется выделять цветом в поле **Описание** отладочного окна
3	Поиск	Осуществляет поиск слова (или других последовательностей символов) в поле **Описание** отладочного окна
4	Установить фильтр	Вызывает окно, с помощью которого осуществляется включение и настройка фильтра. Строки, которые содержат (не содержат) указанные слова должны обязательно быть включены в отладочное окно (или исключены из него)
5	Применить фильтр	Активирует созданный фильтр

9.16 Приложение 2. Создание виртуальных объектов с возможностью задавать события, реакции и состояния

9.16.1 Назначение виртуальных объектов и их реализация в ПК Интеллект

Виртуальные объекты представляют собой программную эмуляцию новых объектов ПК *Интеллект* и позволяют настраивать свои состояния, реакции, события. Работа с виртуальными объектами осуществляется при помощи скриптов, макрокоманд и макрособытий.

Создание виртуальных объектов выполняется с использованием утилит ddi.exe и CustomTypeEditor.exe, расположенных в каталоге <Директория установки ПК *Интеллект*>\Tools.

В разделе Пример создания виртуального объекта рассмотрен пример создания двух типов виртуальных объектов, которые можно использовать, например, для отображения состояния детектора оставленных предметов на карте, или для отображения любых других пользовательских состояний. При этом изменение состояний объекта осуществляется при помощи макрокоманд, скриптов или через IIDK.

Порядок создания и настройки виртуального объекта:

1. Подготовить файл dbi с необходимыми типами объектов.
2. Подготовить файл ddi – в нём указываются события, реакции, состояния и правила перехода состояний для вновь создаваемых объектов.
3. Подготовить файл xml с параметрами вновь создаваемых объектов.
4. Обновить основную Базу данных с помощью Утилиты конвертирования, выбора шаблона и создания резервных копий баз данных idb.exe.
5. Создать виртуальный объект в ПК Интеллект.

9.16.2 Пример создания виртуального объекта

В данном разделе описана последовательность действий для создания следующих виртуальных объектов:

1. Тип CUSTOM, с родительским типом SLAVE (Компьютер).
2. Тип CUSTOM_CHILD, с родительским типом CUSTOM (см. п.1).

Объект типа CUSTOM имеет набор свойств:

1. Параметры custom_param1, custom_param2
2. События: ALARM, INFO, ON, OFF
3. Реакции: ON, OFF
4. Состояния: ON, OFF
5. Машина состояний:
    a. На событие ON выставить состояние ON
    b. На событие OFF выставить состояние OFF

Тип объекта CUSTOM_CHILD создается для демонстрации иерархической структуры и не имеет пользовательских параметров, событий, реакций и состояний.

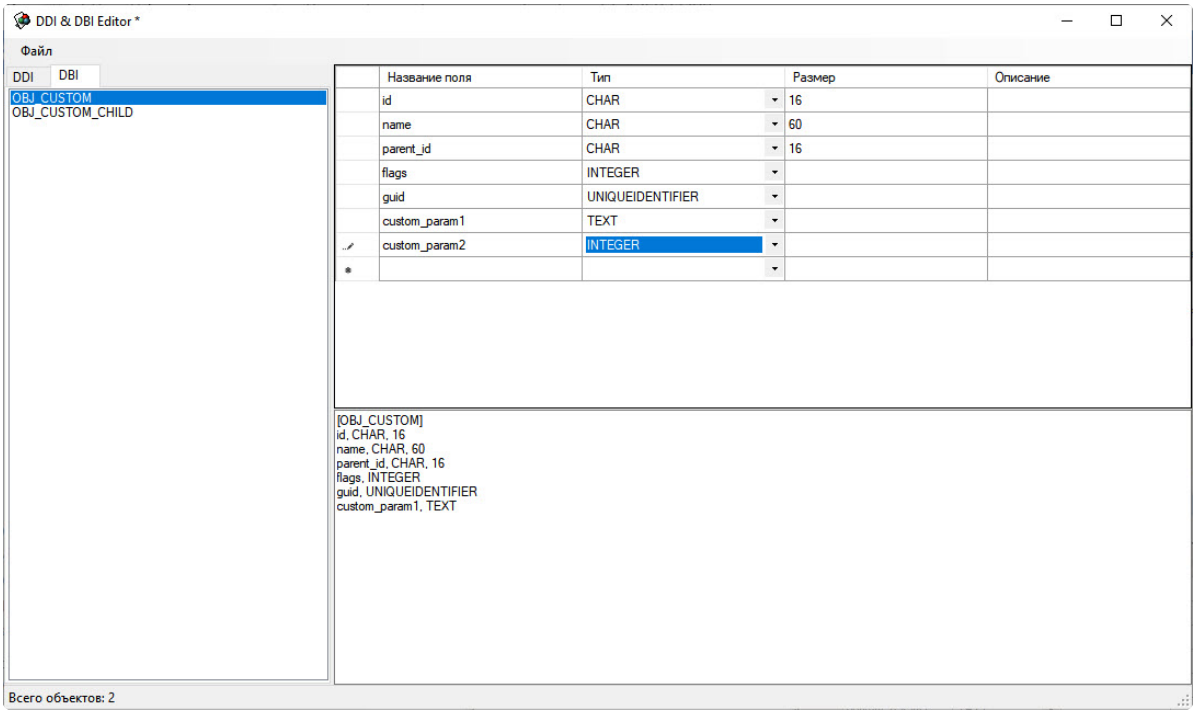
9.16.2.1 Подготовка файла dbi

Подготовка файла dbi осуществляется при помощи утилиты ddi.exe. Работа с ней подробно описана в разделе Утилита редактирования шаблонов баз данных и файла внешних настроек ddi.exe.

Создание файла dbi для типов объектов CUSTOM и CUSTOM_CHILD выполняется в следующей последовательности:

1. Запустить утилиту ddi.exe (см. Утилита редактирования шаблонов баз данных и файла внешних настроек ddi.exe).
2. Перейти на вкладку **DBI**.
3. Создать два объекта OBJ_CUSTOM и OBJ_CUSTOM_CHILD как показано на рисунке ниже.

> **Внимание!**
>
> Названия объектов (таблиц) должны иметь вид OBJ_<тип объекта>.



4. Задать параметры для каждого объекта. Параметры **id**, **name**, **parent_id**, **flags**, **guid** являются обязательными для всех объектов. **Custom_param1**, **custom_param2** в примере на изображении – пользовательские параметры. Также можно задать и прочие параметры, используемые в ПК *Интеллект*. Например, добавление параметра **region_id** позволит задавать для объекта области и разделы (см. Разграничение охраняемого объекта на области и разделы).
5. Сохранить изменения при помощи команды **Сохранить** в меню **Файл**. Сохраненный файл должен иметь расширение dbi и располагаться в директории установки ПК *Интеллект*, например C:\Program Files (x86)\Интеллект\intellect.custom.dbi.

Подготовка файла dbi завершена.

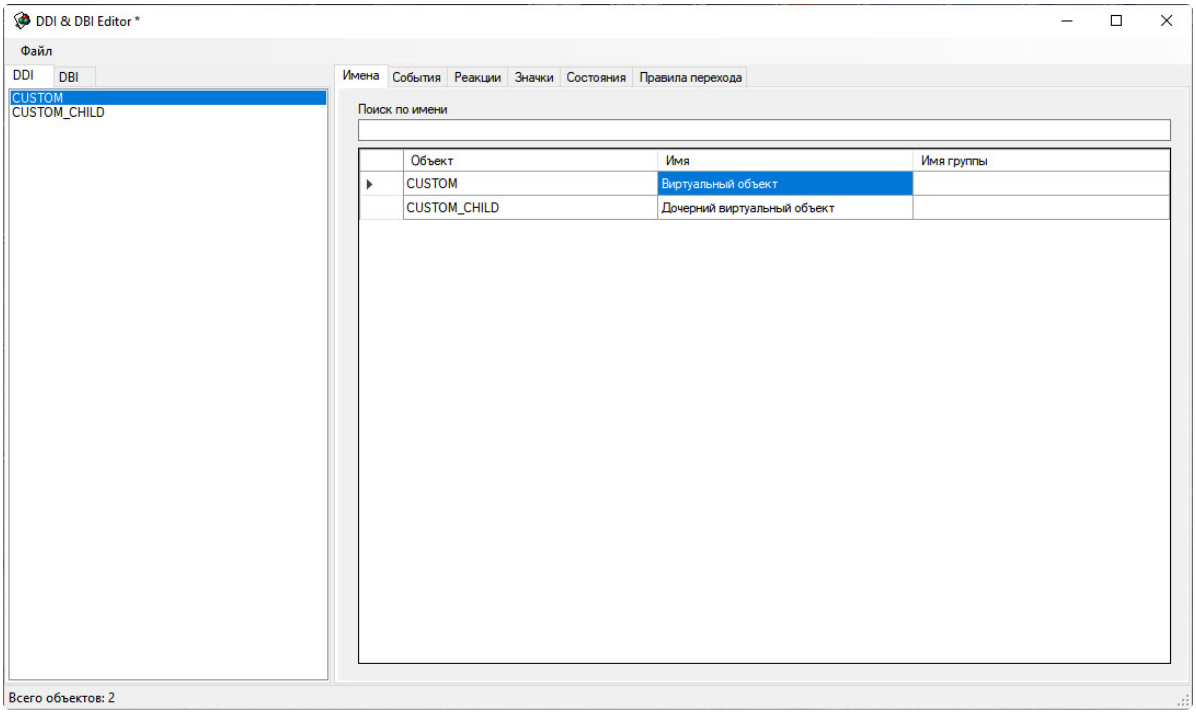
9.16.2.2 Подготовка файла ddi

Подготовка файла ddi осуществляется при помощи утилиты ddi.exe. Работа с ней подробно описана в разделе Утилита редактирования шаблонов баз данных и файла внешних настроек ddi.exe.

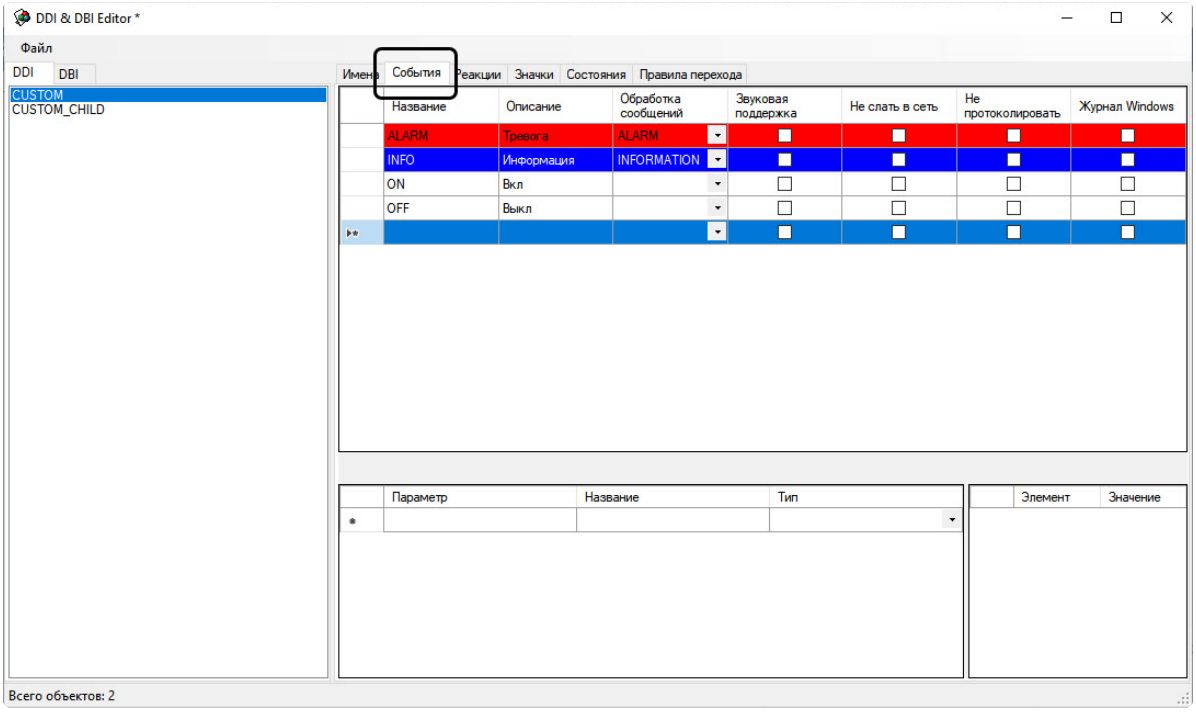
Создание файла ddi для типов объектов CUSTOM и CUSTOM_CHILD выполняется в следующей последовательности:

1. Запустить утилиту ddi.exe (см. Утилита редактирования шаблонов баз данных и файла внешних настроек ddi.exe).

2. На вкладке **DDI** создать два объекта CUSTOM и CUSTOM_CHILD как показано на рисунке ниже.

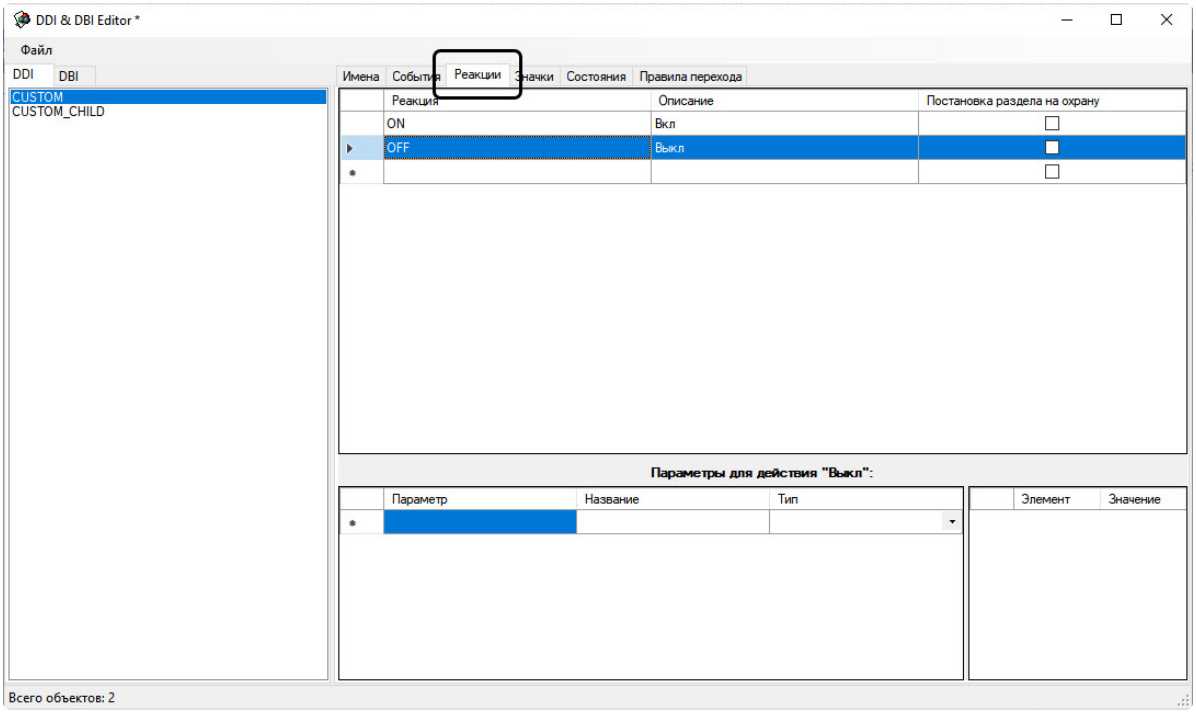


3. Перейти на вкладку **События** и настроить события, которые должны поддерживаться объектом (см. рисунок).

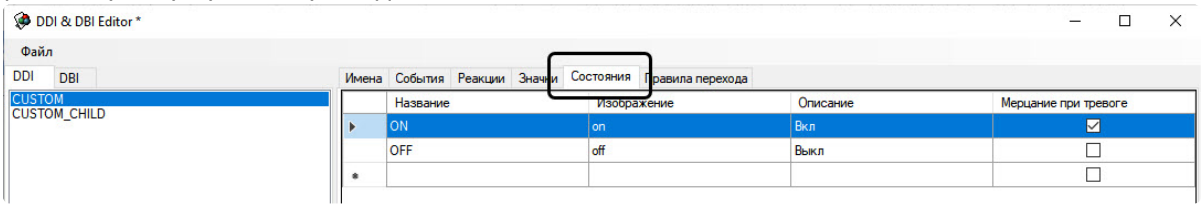


4. Перейти на вкладку **Реакции** и настроить реакции, которые должен поддерживать объект (см. рисунок).

> **Примечание.**
>
> Реакции виртуальных объектов автоматически конвертируются в события. Иными словами, при поступлении реакции виртуальный объект автоматически сгенерирует соответствующее событие.

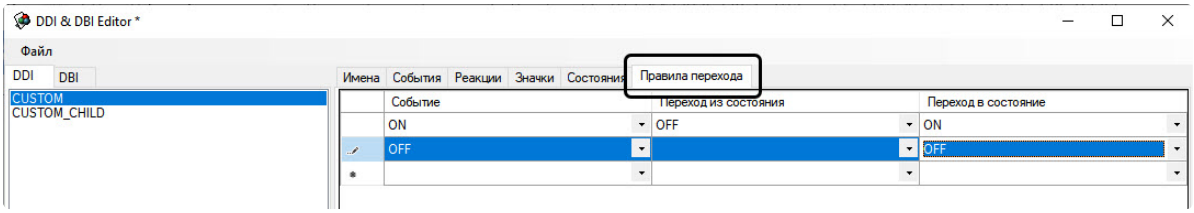


5. Перейти на вкладку **Состояния** и описать состояния, которые может принимать объект. В данном примере рассмотрено два состояния – ON и OFF.



> **Примечание.**
>
> В столбце **Изображение** указывается постфикс имени файла изображения, расположенного в папке <Директория установки ПК *Интеллект*>\Bmp. Например, для объекта CUSTOM это будут файлы custom_off.bmp и custom_on.bmp, соответствующие состояниям ON и OFF. Эти файлы будут использованы модулем Карта.

6. Перейти на вкладку **Правила перехода** и настроить логику изменения состояния объекта.



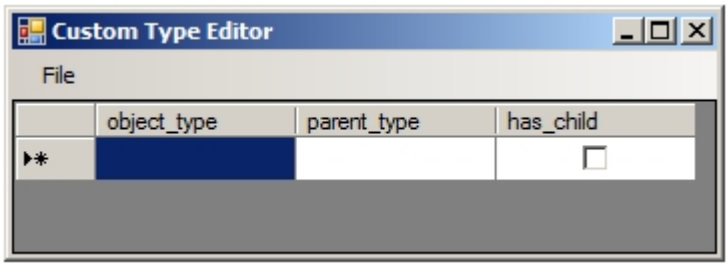
   Правила перехода состояний — это простая машина состояний, входным действием является событие, а результатом — состояние.
   В рассматриваемом примере используется безусловный переход: если пришло событие CUSTOM||ON, производится переход в состояние ON, если пришло событие CUSTOM||OFF — в состояние OFF.

7. Сохранить изменения при помощи команды **Сохранить** в меню **Файл**. Сохраненный файл должен иметь расширение ddi и располагаться в папке, соответствующей требуемому языку, например C:\Program Files (x86)\Интеллект\Languages\ru\intellect.custom.ddi.

Подготовка файла ddi завершена.

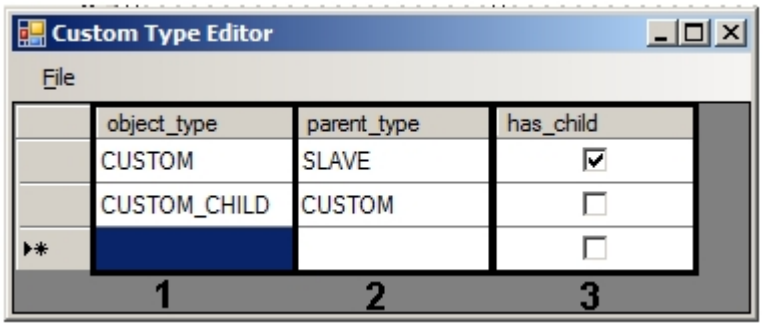
9.16.2.3 Подготовка файла xml

Подготовка файла xml осуществляется при помощи утилиты CustomTypeEditor.exe, расположенной в папке <Директория установки ПК *Интеллект*>\Tools. Общий вид окна утилиты представлен на рисунке.



Создание файла xml для виртуального объекта выполняется следующим образом:

1. В поле **object_type** ввести название типа объекта (1).



2. В поле **parent_type** ввести название родительского типа (2).
3. В случае, если у типа объекта будут иметься дочерние типы, установить флажок в столбце **has_child** (3).

4. Повторить шаги 1-3 для всех типов объектов.
5. Сохранить файл с любым именем и расширением .xml в директории установки ПК *Интеллект* при помощи команды меню **File** – **Save**. Например, для показанного на иллюстрации выше объекта рекомендуется название файла "CUSTOM.xml".

Создание файла xml завершено. Созданный файл будет иметь следующее содержание:

```
<?xml version="1.0" standalone="yes"?>
<objects>
    <object>
        <object_type>CUSTOM</object_type>
        <parent_type>SLAVE</parent_type>
        <has_child>1</has_child>
</object>
    <object>
        <object_type>CUSTOM_CHILD</object_type>
        <parent_type>CUSTOM</parent_type>
    </object>
</objects>
```

При необходимости можно редактировать его и вручную.

В частности, имеется возможность добавить в xml-файл параметр объекта <include_parent_id>1</include_parent_id>. При выставлении значения данного параметра равным 1, идентификаторы дочерних виртуальных объектов будут включать идентификатор родительского объекта. Например, если у объекта CUSTOM есть дочерний объект CUSTOM_CHILD, и идентификатор объекта CUSTOM равен 3, то объекты CUSTOM_CHILD будут создаваться с идентификаторами 3.1, 3.2 и т.д.

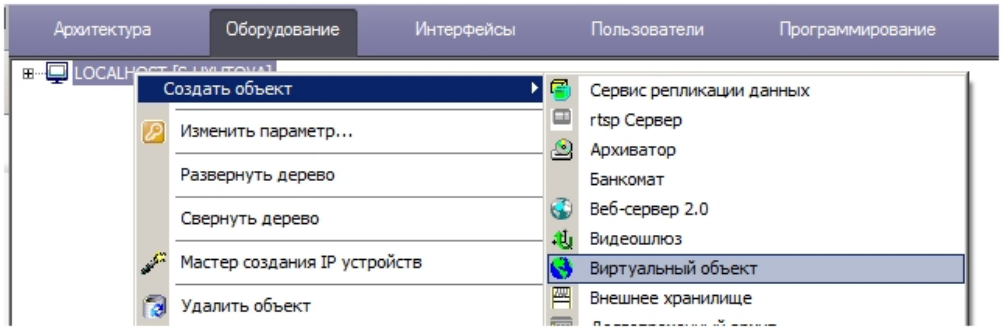
9.16.2.4 Создание и использование виртуального объекта в ПК Интеллект

На странице:

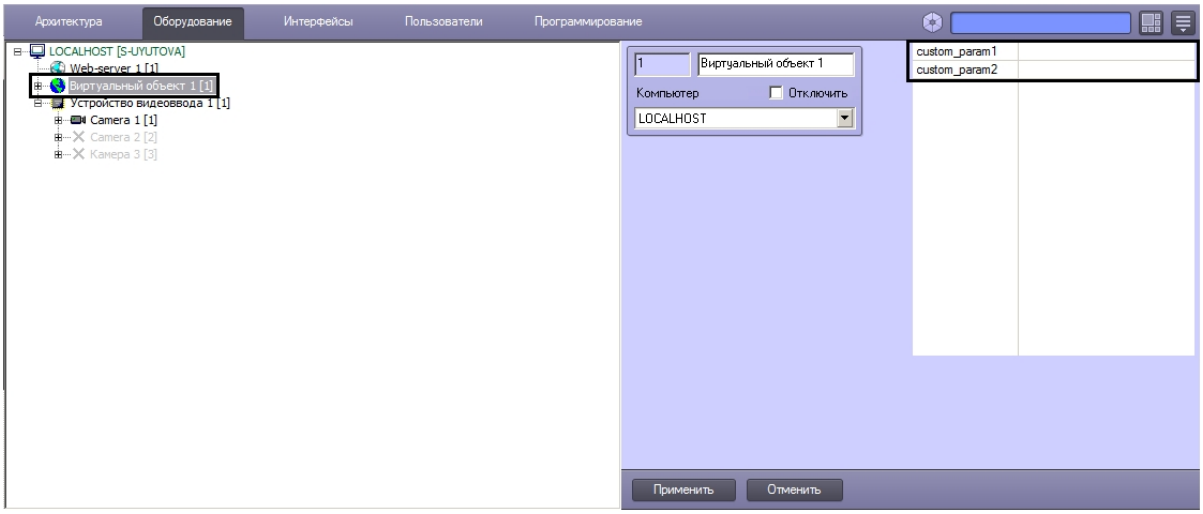
Внимание!

После подготовки необходимых файлов и перед созданием в ПК *Интеллект* виртуальных объектов необходимо обновить основную Базу данных с помощью Утилиты конвертирования, выбора шаблона и создания резервных копий баз данных idb.exe.

После подготовки файлов dbi, ddi и xml появится возможность создать в дереве оборудования ПК *Интеллект* объекты нового типа наряду со стандартными объектами.

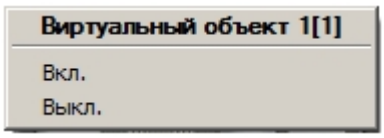


На панели настройки созданного виртуального объекта будут отображены пользовательские параметры – в рассматриваемом примере это custom_param1 и custom_param2. Их значения можно задавать в таблице.



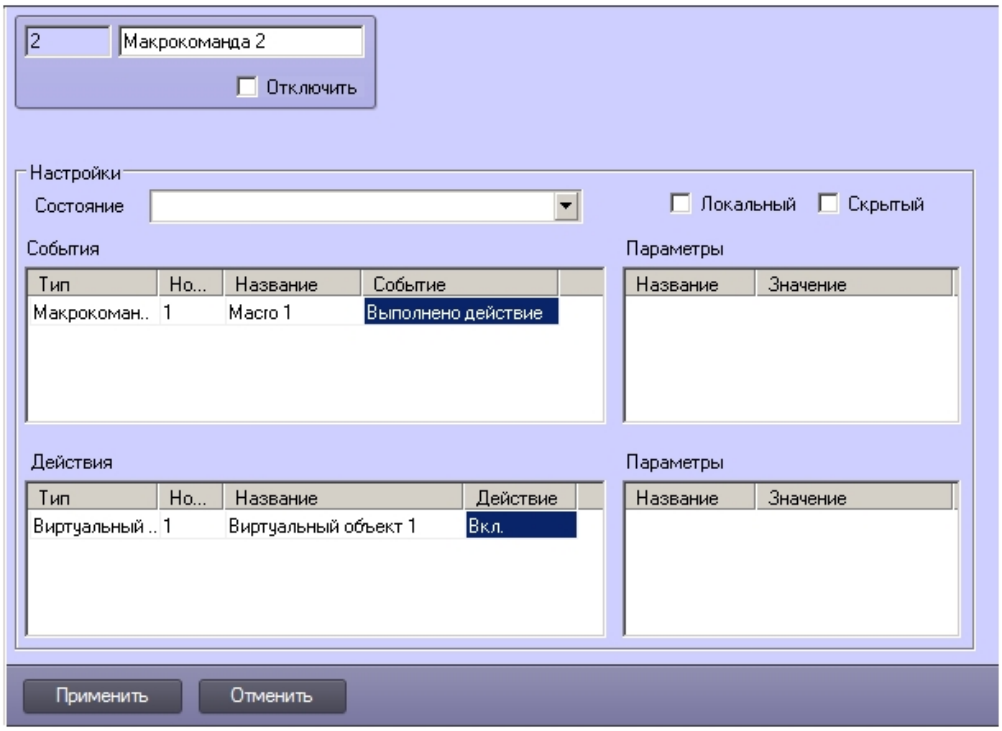
9.16.2.4.1 Отображение на карте

После создания в дереве оборудования объект можно размещать на Карте и выполнять заданные реакции из контекстного меню объекта (см. Настройка интерактивной карты для индикации состояний и управления системными объектами).



9.16.2.4.2 Использование в макрокомандах

После создания виртуального объекта в дереве оборудования имеется также возможность использовать его в макрокомандах.



Примечание.

Реакции виртуальных объектов автоматически конвертируются в события. Таким образом, в рассматриваемом примере, благодаря настроенным правилам перехода состояний (см. Подготовка файла ddi), при выполнении реакции **Вкл.** объект перейдет в соответствующее состояние, и на Карте будет отображена иконка, соответствующая данному состоянию.

9.16.2.4.3 Пример программы на языке JScript для изменения состояния виртуального объекта

Задача. По макрокоманде 1 изменить состояние виртуального объекта с идентификатором 1 на ON и отобразить на карте соответствующую данному состоянию иконку.

Решение. Поскольку в рассматриваемом примере виртуального объекта настроены правила перехода состояний, при отправке события ON от виртуального объекта его состояние будет автоматические изменено на ON, а на карте будет отображена иконка, указанная в файле ddi для данного состояния (см. Подготовка файла ddi). Скрипт для отправки события ON имеет вид:

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")

{
    var msgevent = CreateMsg();

    msgevent.SourceType = "CUSTOM";

    msgevent.SourceId = "1";

    msgevent.Action = "ON";

    NotifyEvent(msgevent);

}
```

10 Описание событий и реакций объектов системы

В данной главе указаны события, команды и реакции для основных объектов системы.

В ПК *Интеллект* выделяют два типа системных сообщений: события и реакции. События, как правило, несут только информационную нагрузку и используются для рассылки оповещений по всем ядрам ПК *Интеллект*, соединенным между собой при конфигурировании архитектуры. Под реакциями понимаются команды, отправляемые конкретным системным объектам. Реакции передаются непосредственно тем ядрам, на которых зарегистрирован объект, а не всей системе. Для генерации реакций используются методы DoReactStr и DoReact. Для генерации событий – NotifyEventStr и NotifyEvent.

Примечание.

События для объектов системы можно просмотреть одним из следующих способов:

1. Просмотр содержимого файла intellect.ddi в утилите «ddi.exe» (см. Получение списка системных названий объектов, реакций и событий ПК Интеллект).
2. Просмотр событий для выбранного объекта системы на панели настроек системного объекта **Макрокоманда** (см. Создание и использование макрокоманд).

.

10.1 GRABBER Устройство видеоввода

Объект **GRABBER** соответствует системному объекту **Устройство видеоввода**.

От объекта **GRABBER** поступают события, представленные в таблице. Запуск процедуры происходит при возникновении соответствующего события.

Описание событий от объекта **GRABBER**:

События	**Описание события**
+12V	Ошибка напряжения +12V
+3.3V	Ошибка напряжения +3.3V
+5V	Ошибка напряжения +5V
-12	Ошибка напряжения -12V

События	**Описание события**
-5V	Ошибка напряжения -5V
CPU_FAN	Количество оборотов вентилятора
CPU_TEMP	Температура процессора
SYS_TEMP	Температура чипсета MB
UPS_COMMLOST	Потеря связи
UPS_FATAL_ERROR	Ошибка подключения
UPS_LOWBATT	Села батарея
UPS_ONBATT	Переход на питание от батареи
UPS_ONLINE	Восстановление питания от сети
UPS_REPLACEBATT	Требуется замена батареи
UPS_SHUTTING	Выключение
VCORE	Напряжение ядра процессора
AUDIO_SIG_LOST	Потеря звука
CONNECT_FAIL	Ошибка подключения
CONNECT_OK	Подключено
NETWORK_FAILURE	Соединение потеряно
STATE_CONNECTED	Соединение восстановлено

Список команд и параметров для объекта **GRABBER** представлен в таблице:

Команда – описание команды	**Параметры**	**Описание параметров**
"SETUP" – устанавливает параметры устройства видеоввода	chan<>	Номер PCI слота (0,1,2,...,32)
	mode<>	Скорость граббера/оцифровки (0 – максимальная, 1 – средняя, 2 – минимальная)
	resolution<>	Разрешение: 0 – стандартное, четверть кадра (384x288); 1 – высокое, полукадр (768x288); 2 – максимальное, кадр (768x576)
	format<>	Формат видеосигнала (PAL, NTSC)
	drives<>	Диски для записи видеоархива (DRIVE1:\, DRIVE2:\ ... DRIVEN:\)
	cams<>	Количество подключенных видеокамер
	auth<>	Данные авторизации
	ip<>	IP-адрес сетевой платы видеоввода
	name<>	Имя объекта
	flags <>	Флаги
	ip_port<>	IP-порт
	password<>	Пароль
	type<>	Тип оцифровки
	username<>	Логин
	watchdog<>	Включение WatchDog (0 – выключен, 1 – включен)

Команда – описание команды	**Параметры**	**Описание параметров**
"SET_DRIVES" – устанавливает диски для записи видеоархива	drives<>	Диски для записи видеоархива
"MUX1_OFF" – отключить вывод видео через аналоговый выход 1	-	-
"MUX2_OFF" – отключить вывод видео через аналоговый выход 2	-	-
"MUX3_OFF" – отключить вывод видео через аналоговый выход 3	-	-
"SET_IPINT_PARAM" – Установить (изменить) параметры IP-устройства. Реакция позволяет менять настройки IP-устройства, не заходя в его web-интерфейс. *Примечание. Для работы реакции необходимо включить режим многопоточного видеосигнала (см.* Руководство администратора*, раздел* Настройка многопоточного видеосигнала*), а также* Приложение 2. Определение значений param_id и param_value для реакции SET_IPINT_PARAM	param_id<>	Название параметра. Набор параметров для каждой камеры индивидуален – см. Приложение 2. Определение значений param_id и param_value для реакции SET_IPINT_PARAM
	param_value<>	Значение параметра. Набор параметров для каждой камеры индивидуален – см. Приложение 2. Определение значений param_id и param_value для реакции SET_IPINT_PARAM
	cam_id<>	Идентификатор камеры в ПК *Интеллект*
	vstream_id<>	Номер видеопотока (не обязательный параметр). Имеет вид "Номер камеры"."Номер потока", например 1.1, 1.2.
"START" – начать проигрывание видеоролика в виртуальном устройстве видеоввода	-	-
"STOP" – остановить проигрывание видеоролика в виртуальном устройстве видеоввода	-	-

Команда – описание команды	**Параметры**	**Описание параметров**
"ENABLE" – включить (снять флажок **Отключить** на панели настройки объекта)	recursive<>	Возможные значения параметра: 0 – включить только **Устройство видеоввода** 1 – включить **Устройство видеоввода** и все объекты **Камера**, созданные на базе него
"DISABLE" – отключить (установить флажок **Отключить** на панели настройки объекта)	recursive<>	Возможные значения параметра: 0 – отключить только **Устройство видеоввода** 1 – отключить **Устройство видеоввода** и все объекты **Камера**, созданные на базе него

Свойства объекта **GRABBER** показаны в таблице:

Свойства объекта GRABBER	**Описание свойств объекта**
ID<>	Идентификатор объекта
PARENT_ID<>	Номер устройства видеоввода

10.2 CAM Камера

Объект **CAM** соответствует системному объекту **Камера**.

От объекта **CAM** поступают события, представленные в таблице. Запуск процедур происходит при возникновении соответствующего события.

Описание событий от объекта **CAM**:

События	**Описание событий**	**Комментарий**
ARM	Камера поставлена на охрану	Если постановка на охрану выполнена Оператором с **Карты** или из **Монитора видеонаблюдения**, в параметре user_id<> содержится идентификатор пользователя, выполнившего это действие

События	**Описание событий**	**Комментарий**
ATTACH	Подключение	
BLINDING	Камера залеплена	
DETACH	Обрыв	Событие генерируется при потере входного сигнала с камеры на плате видеоввода
DISARM	Камера снята с охраны	Если снятие с охраны выполнено Оператором с **Карты** или из **Монитора видеонаблюдения**, в параметре user_id<> содержится идентификатор пользователя, выполнившего это действие
FILE_REC_ERROR	Ошибка записи на диск	Событие генерируется, когда происходит ошибка записи видеоархива на диск
MD_START	Тревога	
MD_STOP	Конец тревоги	
PRINT	Печать кадра	
REC	Запись на диск	Если запись инициирована Оператором с **Карты** или из **Монитора видеонаблюдения**, в параметре user_id<> содержится идентификатор пользователя, выполнившего это действие
REC_STOP	Остановка записи на диск	Если запись остановил Оператор с **Карты** или из **Монитора видеонаблюдения**, в параметре user_id<> содержится идентификатор пользователя, выполнившего это действие
UNBLINDING	Камера открыта	
RECORDER_ON	Запись включена	

События	**Описание событий**	**Комментарий**
RECORDER_OFF	Запись выключена	
DISC_MOUNT	Диск подмонтирован	
DISC_UNMOUNT	Диск отмонтирован	
FINISHED_AVI_EXPORT	Экспорт видео завершен	В случае, если попытка экспорта видео завершилась неудачей, событие имеет ненулевой параметр error_result. В параметре param<0> содержится дополнительная информация, отображаемая в соответствующем столбце в **Протоколе событий** в формате "ИмяКомпьютера;ПериодЭкспорта;ИмяПользователя;ИдентификаторПользователя" – например, param0<LOCALHOST;04-06-18 16:50:55_04-06-18 16:55:55;;1>
MD_LIMIT	Превышено количество объектов в кадре	Событие генерируется при превышении количества обнаруженных трекером объектов в кадре (см. Создание и настройка объекта Трекер). Событие генерируется при каждом изменении количества объектов (в большую или меньшую сторону), пока оно не будет меньше пороговой. В параметре object_count<> передается число объектов в кадре, которое обнаружено трекером, превышает заданный лимит и отличается от предыдущего значения. См. также описание события NEW_OBJECT ниже
NEW_OBJECT	Трекер обнаружил новый объект в кадре	Помимо прочих, содержит следующие параметры: • total<> – общее количество объектов в кадре на момент возникновения события • new_id<> – идентификатор обнаруженного объекта
ARCH_DELETE	Удаление записи	Удаление записи архива по камере из **Монитора видеонаблюдения**

События	**Описание событий**	**Комментарий**
OPEN_FILE	Открытие файла	Открытие файла для воспроизведения виртуальным устройством видеоввода. Свидетельствует о начале воспроизведения следующего файла в выбранной папке. Помимо прочих, содержит следующие параметры: • name<> – название проигрываемого файла • tss<> – время в формате UTC в миллисекундах с 01.01.1970
CLOSE_FILE	Закрытие файла	Завершение воспроизведения файла виртуальным устройством видеоввода. Свидетельствует об окончании воспроизведения файла в выбранной папке. Помимо прочих, содержит следующие параметры: • name<> – название файла, проигрывание которого завершилось • tss<> – время в формате UTC в миллисекундах с 01.01.1970
ARCH_PROTECTED	Файл защищен от перезаписи	Событие отображается, когда пользователь защищает файл от перезаписи. В параметре param<0> содержится дополнительная информация об имени компьютера, фрагменте, имени и id пользователя, отображаемая в столбце **Доп. инфо** в **Протоколе событий** в формате "ИмяКомпьютера;ЗащищенныйПериод;ИмяПользователя;ИдентификаторПользователя" – например, param0<LOCALHOST;04-06-18 16:50:55.612_04-06-18 16:55:55.612;User 1;1>
ARCH_UNPROTECTED	Снята защита файла от перезаписи	Событие отображается, когда пользователь снимает защиту файла от перезаписи. В параметре param<0> содержится дополнительная информация об имени компьютера, фрагменте, имени и id пользователя, отображаемая в столбце **Доп. инфо** в **Протоколе событий** в формате "ИмяКомпьютера;ЗащищенныйПериод;ИмяПользователя;ИдентификаторПользователя" – например, param0<LOCALHOST;04-06-18 16:50:55.612_04-06-18 16:55:55.612;User 1;1>

События	**Описание событий**	**Комментарий**
FRAME_SKIPPED	Пропуски кадров	Событие FRAME_SKIPPED поступает в ПК *Интеллект*, если есть пропуски кадров при записи в архив. По умолчанию событие генерируется, если за период проверки было более 50 пропусков кадров. Когда пропуски кадров прекращаются, генерируется событие FRAME_SKIPPED_STOP. В описании этого события содержится информация о количестве пропущенных кадров и промежутке времени, в который были пропуски. Оба события по умолчанию генерируются не чаще чем раз в 30 секунд. События регулируются следующими ключами реестра (см. Справочник ключей реестра): • событие FRAME_SKIPPED можно отключить с помощью ключа реестра FileSystem.NotifyCoreFrameSkipped; • время задержки между сменой состояний в секундах задаётся в ключе FileSystem.RecordingStateChangeDelay. Минимальное значение – 30 секунд. Если указать меньшее значение, то всё равно будет использоваться минимальное – 30 (ключ в реестре перезапишется); • количество пропущенных кадров за период, при котором будет генерироваться FRAME_SKIPPED, задаётся в ключе FileSystem.MaxSkippedFramesByPeriod.
FRAME_SKIPPED_STOP	Конец пропуска кадров	
ARCH_BOOKMARKED	Создана закладка	Событие отображается, когда пользователь добавляет закладку. В параметре param<0> (столбце **Доп. инфо** в **Протоколе событий**) содержится комментарий к закладке
ARCH_UNBOOKMARKED	Удалена закладка	Событие отображается, когда пользователь удаляет закладку
TEMPERATURE_ALARM	Порог температуры	В параметре param0<> содержится значение температуры, полученное от тепловизора
IGNORE_KEEP_NO_LESS	Игнорирование "Хранить не менее"	Событие генерируется при удалении из архива видеозаписей, которые должны были бы храниться в течение более длительного периода времени, задаваемого параметром **Хранить не менее**. См. также Настройка глубины архива по видеокамере

События	**Описание событий**	**Комментарий**
AVAILABILITY	Наличие ключевой позиции в файле лицензии и количество объектов	Событие приходит в ответ на команду GET_AVAILABILITY и содержит информацию о количестве добавленных в ключ лицензии объектов для заданной позиции. Параметры: • qty_localpriority<> – количество разрешенных объектов заданного типа на этом компьютере (локальная часть ключа) • qty<> – общее количество разрешенных объектов заданного типа в лицензии (общая часть ключа) • pos<> – номер позиции в ключе лицензии Если позиция в ключе отсутствует, то событие будет со значением ноль
WRITING_FAILED	Ошибка записи	
FILESYSTEM_FAILED	Ошибка файловой системы	Содержит обязательный параметр error_msg<> – текст ошибки; также может содержать параметры cam<> – идентификатор камеры и error_code<> – код ошибки. Пример: CAM|-1|FILESYSTEM_FAILED|error_msg<Failed to delete index file: D:\\\VIDEO\\\INDEX\\\14061608.idx, error code: 5.>,error_code<5>

Список команд и параметров для объекта **CAM** представлен в таблице.

Команда – описание команды	**Параметры**	**Описание параметров**
"SETUP" – устанавливает (изменяет) параметры камеры	rec_priority<>	Приоритет записи (от 0 до 3, 0 – обычный, 3 – все ресурсы)
	compression<>	Степень компрессии (0 – компрессия отсутствует, 1-макс. качество, ..., 5 – мин. качество)
	sat_u<>	Насыщенность цвета (0 – мин, 10 – макс)
	proc_time<>	Период дозаписи (0-30 сек)
	manual<>	Управление настройкой яркости и контрастности (0 – ручное; 1 – автоматическое; 2 – автоматическое, но около значений, выставленных вручную)

Команда – описание команды	**Параметры**	**Описание параметров**
	contrast<>	Контрастность (0 – мин, 10 – макс)
	md_size<>	Размер объектов детектора движения (1-16)
	md_mode<>	Режим записи пауз (1 – включено, 0 – выключено)
	audio_type<>	Тип звукового сопровождения
	pre_rec_time<>	Время предзаписи (0-20 сек)
	bright<>	Яркость (0 – мин, 10 – макс)
	audio_id<>	Номер микрофона (пустой параметр, если нет микрофона)
	rec_time<>	Скорость записи (1-30 кадров/сек, 0 – не используется)
	alarm_rec<>	Запись тревог (1 – включено, 0 – выключено)
	hot_rec_time<>	Время горячей записи (0-30 сек)
	hot_rec_period<>	Период горячей записи (0-20 сек)
	mux<>	Номер канала (0 – 1 канал, 15 – 16 канал)
	color<>	Цвет (0 – черно-белый, 1 – цветной)
	activity<>	-
	arch_days<>	Количество дней архива

Команда – описание команды	**Параметры**	**Описание параметров**
	blinding<>	Камера залеплена
	config_id<>	-
	decoder<>	-
	flags<>	Флаги
	fps<>	Скорость записи (0 – не используется, 1 – 30 кадров/сек)
	ifreq<>	Частота опорных кадров в последовательности (1 – каждый кадр опорный, 2 – 100 кадр)
	mask 0, mask1, mask2, mask3, mask4	Маска детектора
	md_contrast<>	Чувствительность детектора движения (0-15)
	motion<>	Оценка движения компрессора (5-255)
	name<>	Имя объекта
	password_crc<>	Пароль на видеоархив
	priority<>	Приоритет источника постановки на запись (0 – автоматическое, 1 – ручное)
	resolution<>	Разрешение (0 – стандартное CIF, 1 – высокое 2CIF, 2 – максимальное 4CIF)
	type<>	Тип объекта

Команда – описание команды	**Параметры**	**Описание параметров**
	yuv<>	Цветовая схема кодирования видеосигнала (0 – YUV4:2:0, 1 – YUV4:2:2)
"DELETE" – отключает камеру	-	-
"START_VIDEO" – включает видеопоток для текущей камеры	slave_id<>	Имя компьютера, к которому подключена камера
	comress<>	Степень компрессии
	register_only<>	-
"STOP_VIDEO" – выключает видеопоток для текущей камеры	slave_id<>	Имя компьютера, к которому подключена камера
"REQUEST_MASK"	mask<>	Маска
"MUX1", "MUX2", "MUX3" – вывести изображение камеры на 1, 2, 3 аналоговые выходы	-	-
"ACTIVATE" – вывести камеру на монитор	monitor<>	Номер монитора
"ARM" – поставить камеру на охрану	-	-
"DISARM" – снять камеру с охраны	-	-
"REC" – начать запись камеры	time<>	Время записи в секундах, если равно нулю, то записывается 1 кадр
	rollback<>	Если равно 1, то запись производится с откатом

Команда – описание команды	**Параметры**	**Описание параметров**
	priority<>	Задает приоритет команды начала записи. Подробнее см. Приложение 1. Приоритеты команд начала и остановки записи
	stream_id<>	Задает номер потока для записи. Номер потока имеет вид "n.m", где n – id камеры, m – номер потока. *Примечание. Если указанный поток не используется ни по какому другому назначению, кроме записи по команде (и по выбору на клиенте), необходимо убедиться, что для него не установлен флажок* ***Блокировать отключение неиспользуемых потоков*** *– см.* Панель настройки объекта Камера
"REC_STOP" – остановить запись камеры	priority<>	Задает приоритет команды остановки записи. Подробнее см. Приложение 1. Приоритеты команд начала и остановки записи
	user_id<>	Если остановка записи была выполнена пользователем из **Монитора видеонаблюдения**, то в данном параметре передается идентификатор пользователя. В противном случае данный параметр отсутствует
	from_macro<>	Если остановка записи была выполнена по макрокоманде, то в данном параметре передается идентификатор макрокоманды. В противном случае данный параметр отсутствует
"SET_MASK" – установить маску	mask<>	Маска
"ADD_SUBTITLES" – добавить титры	command<>	Текст накладываемых титров
	title_id<>	Идентификатор объекта **Титрователь**, используемого для наложения титров

Команда – описание команды	**Параметры**	**Описание параметров**
	page<>	Обязательный параметр при записи титров в базу данных титров для обеспечения в дальнейшем возможности поиска по титрам. Возможные значения: BEGIN (начало записи в базе), END (конец записи в базе)
"SIP_CONNECT" – Sip подключен	-	-
"SIP_DISCONNECT" – Sip отключен	-	-
"SET_IPINT_PARAM" – Установить (изменить) параметры IP-устройства. Реакция позволяет менять настройки IP-устройства не заходя в его web-интерфейс. *Примечание. Для работы реакции необходимо включить режим многопоточного видеосигнала – см.* Настройка многопоточного видеосигнала*, а также* Приложение 2. Определение значений param_id и param_value для реакции SET_IPINT_PARAM	param_id<>	Название параметра. Набор параметров для каждой камеры индивидуален – см. Приложение 2. Определение значений param_id и param_value для реакции SET_IPINT_PARAM
	param_value<>	Значение параметра. Набор параметров для каждой камеры индивидуален – см. Приложение 2. Определение значений param_id и param_value для реакции SET_IPINT_PARAM
	vstream_id<>	Номер видеопотока (необязательный параметр). Имеет вид "Номер камеры"."Номер потока", например 1.1, 1.2
"GET_FRAME" – получить кадр с камеры, даже если она не отображается на **Мониторе видеонаблюдения**	path<>	Путь для сохранения кадра. Если параметр отсутствует, в системе будет сформировано событие FRAME_SENT с параметром data. Обработка данного события описана в разделе Метод SaveToFile

Команда – описание команды	**Параметры**	**Описание параметров**
	stream<>	Необязательный параметр. Задает поток в ПК *Интеллект*, с которого требуется получить кадр. Поток можно указать по номеру или по назначению. Возможные назначения: • stream_archive – поток для записи в архив • stream_alarm – поток для записи в архив по тревогам • stream_client – поток для отображения • stream_analytic – поток для видеоаналитики Номер потока состоит из идентификатора камеры и номера потока, например, 4.3 – третий поток с камеры 4
	time<>	Необязательный параметр. Указывается, если требуется запросить кадр архива. Формат значения параметра: ДД-ММ-ГГГГ ЧЧ:ММ:СС – например, time<19-09-2017 11:35:34>
	gate<>	Необязательный параметр. Задает сетевое имя **Видеошлюза**, из архива которого следует получать кадр
	arch<>	Необязательный параметр. Задает сетевое имя **Долговременного архива**, из которого следует получать кадр
	slave_id<>	Необязательный параметр. Задает сетевое имя Сервера, из архива которого следует получать кадр
	password_crc<>	Необязательный параметр. Задает CRC-последовательность, которая будет записана в результирующий файл, содержащий запрошенный кадр
"ARCH_DEL_RECORD" – удалить записи архива за определенный период	fromTime<>	Обязательный параметр. Время в формате ГГГГ-ММ-ДДТЧЧ:ММ:СС.НННН, где НННН – миллисекунды. Будут удалены записи, начиная с первой, содержащей указанный момент времени, и заканчивая последней, содержащей момент времени toTime. Если не указано время в параметре toTime, будет удалена только одна запись

Команда – описание команды	**Параметры**	**Описание параметров**
	toTime<>	Необязательный параметр. Время в формате ГГГГ-ММ-ДДТЧЧ:ММ:СС.НННН, где НННН – миллисекунды. Описание см. выше
"REC_RESTART" – перезапустить запись по камере	-	-
"ARCH_BOOKMARK_RECORD" – создать закладку	time1<>	Дата начала периода архива, включенного в закладку, в формате ДД-ММ-ГГ ЧЧ:ММ:СС.НННН, где НННН – миллисекунды
	time2<>	Дата окончания периода архива, включенного в закладку, в формате ДД-ММ-ГГ ЧЧ:ММ:СС.НННН, где НННН – миллисекунды
	comment<>	Комментарий к закладке
	slave_id<>	Идентификаторы компьютера и **Монитора видеонаблюдения**, с использованием которых будет выполняться создание закладки. Формат параметра: <имя компьютера>.<айди монитора>. Например, slave_id<WS2.1> – здесь WS2 является идентификатором компьютера, а 1 – идентификатором Монитора видеонаблюдения
"CRUISE_START" – автопанорамирование	cruise_id<>	Номер маршрута на камере
	action<>	Выполняемое действие: • CRUISE_START – начать автопанорамирование по указанному маршруту • PATROL_PLAY – начать патрулирование по указанному маршруту
	cam_id<>	Идентификатор камеры

Команда – описание команды	**Параметры**	**Описание параметров**
"GET_DEPTH" – получить глубину архива. В ответ на реакцию в системе формируется событие ARCHIVE_DEPTH от объекта SLAVE (см. SLAVE Компьютер). Отсутствие одного или обоих параметров означает запрос глубины по записям для всех возможных значений параметра	drive<>	Диск или сетевой путь, по которому запрашивается глубина архива. Название диска задается формате "<буква диска>:\\", например drive<D:\\> *Примечание. Символ "\" – экранируемый.* Сетевой путь задается в формате UNC
	arch	Указывает, что требуется запросить глубину долговременного архива. Пример. `DoReactStr("CAM","2","GET_DEPTH","drive<D:\\>,cam<2>,arch");`
	gate	Указывает, что требуется запросить глубину архива видеошлюза. Пример. `DoReactStr("CAM","1","GET_DEPTH","drive<V:\\>,gate");`
"CLEAR_SUBTITLES" – удалить все титры с видеоизображения	title_id<>	Идентификатор объекта **Титрователь**
"GET_AVAILABILITY" – проверить наличие ключевой позиции	pos<>	Номер позиции в ключе лицензии

Свойства объекта **CAM** показаны в таблице:

Свойства объекта CAM	**Описание свойств объекта**
ID<>	Идентификатор объекта
PARENT_ID<>	Идентификатор родительского объекта
TELEMETRY_ID<>	Идентификатор модуля телеметрии (ID поворотника)

REGION_ID<>	Идентификатор региона

Объект **CAM** может находиться в состояниях, описанных в таблице:

Состояние объекта CAM	**Описание состояния**
"ALARMED"	Камера находится в тревожном состоянии
"DISARM_DETACHED"	Нет сигнала от камеры
"DETACHED"	Нет сигнала от камеры
"ARMED"	Камера поставлена на охрану
"DISARMED"	Камера снята с охраны

10.3 MONITOR Монитор видеонаблюдения

Объект **MONITOR** соответствует системному объекту **Монитор**.

От объекта **MONITOR** поступают события, представленные в таблице. Запуск процедур происходит при возникновении соответствующего события.

Описание событий от объекта **MONITOR**:

Событие	**Описание события**	**Комментарий**
STARTED_AVI_EXPORT	Начало экспорта видео	Среди прочих, событие содержит следующие параметры: • slave_id<> – оператор, запустивший экспорт • param1<> – номер камеры, по которой осуществляется экспорт, а также дата и время начала периода экспорта. Параметр принимает значение вида "<№записи> Камера <id> (дд-мм-гг чч:мм:сс)", например param1<01 Камера 1 (05-10-17 10:23:21)> • time<> – время начала экспорта

FINISHED_AVI_EXPORT	Конец экспорта видео	Среди прочих, событие содержит следующие параметры: • slave_id<> – оператор, запустивший экспорт • param1<> – номер камеры, по которой осуществляется экспорт, а также дата и время окончания периода экспорта. Параметр принимает значение вида "<№записи> Камера <id> (дд-мм-гг чч:мм:сс)", например param1<01 Камера 1 (05-10-17 10:40:21)> • time<> – время окончания экспорта • param<0> – дополнительная информация, отображаемая в соответствующем столбце в Протоколе событий, в формате: "ИмяКомпьютера;ПериодЭкспорта;ИмяПользователя;ИдентификаторПользователя", например, param0<LOCALHOST;04-06-18 16:50:55_04-06-18 16:55:55;;1>
AVI_EXPORT_RESULT	Результат экспорта видео	Событие имеет те же параметры, что и START_AVI_EXPORT, с добавлением параметра error_result<>, принимающего одно из следующих значений: 0 – экспорт успешно выполнен 1 – неизвестно 2 – задача занята 3 – не готово 4 – неверный интервал 5 – ошибка файла
PLAY_START	Начало проигрывания фрагмента архива	
PLAY_STOP	Конец проигрывания фрагмента архива	
INTERFACE_MANIPULATION	Изменение визуализации	param<0> – дополнительная информация, отображаемая в соответствующем столбце в Протоколе событий, содержит идентификатор перемещенной по раскладке камеры
LAYOUT_DEL	Удаление раскладки	param<0> – дополнительная информация, отображаемая в соответствующем столбце в Протоколе событий, содержит имя удаленной раскладки

LAYOUT_ADD	Добавление раскладки	param<0> – дополнительная информация, отображаемая в соответствующем столбце в Протоколе событий, содержит имя добавленной раскладки
LAYOUT_ACTIVATE	Смена активной раскладки	param<0> – дополнительная информация, отображаемая в соответствующем столбце в Протоколе событий, содержит имя активированной раскладки
REPLACE_CAM	Смена положения камеры	param<0> – дополнительная информация, отображаемая в соответствующем столбце в Протоколе событий, в формате: <Имя камеры 1> → <Имя камеры 2>
ACTIVATE_CAM	Активация камеры	auto_switch<> – указывает, было ли включено автоматическое листание во время активации камеры. Можно использовать для отключения автоматического листания при активации окна видеонаблюдения
CAM_EXPAND	Увеличение Окна видеонаблюдения на весь Монитор	Событие генерируется только если установлены следующие ключи реестра (см. Справочник ключей реестра): • MaximizeCameraOnDblClk=1 • MinimizeCameraOnDblClk=1 Параметры события: • param0<> – идентификатор камеры • user_id<> – идентификатор пользователя, выполнившего действие
CAM_COLLAPSE	Уменьшение Окна видеонаблюдения	Событие генерируется только если установлены следующие ключи реестра (см. Справочник ключей реестра): • MaximizeCameraOnDblClk=1 • MinimizeCameraOnDblClk=1 Параметры события: • param0<> – идентификатор камеры • user_id<> – идентификатор пользователя, выполнившего действие

Список команд и параметров для объекта **MONITOR** представлен в таблице:

Команда – описание команды	**Параметры**	**Описание параметров**
"REMOVE" – удаляет камеру с монитора	cam<>	ID камеры в дереве настроек, которую необходимо удалить с монитора

Команда – описание команды	**Параметры**	**Описание параметров**
	show<>	Необязательный параметр. Возможные значения: • 0 – не обновлять раскладку в Мониторе после удаления камеры. Может оставаться пустое пространство, не занятое Окнами видеонаблюдения • 1 – обновлять раскладку в Мониторе после удаления камеры, чтобы минимизировать пустое пространство
"REMOVE_ALL" – удаляет все камеры с монитора	-	-
"STOP_VIDEO" – останавливает видеопоток камеры	cam<>	ID камеры в дереве настроек, видеопоток от которой необходимо остановить
"REPLACE" – удаляет все камеры с монитора и вызывает указанную камеру	slave_id<>	Имя компьютера, которому принадлежит монитор, в скрипте можно подставить owner
	cam<>	ID камеры в дереве настроек, которую необходимо вывести на монитор
	name<>	Название камеры, которое будет отображаться в левом нижнем углу
	audio_type<>	-
	audio_id<>	-
	arch_id<>	-
	control<>	0 – только просмотр архива, 1 – также возможно и управление (постановка/снятие с охраны, запись)

Команда – описание команды	**Параметры**	**Описание параметров**
"ADD_SHOW" – добавляет камеры на монитор *Примечание. См. также PLACE_CAM_IN_LAYOUT_CELL*	cam<>	ID камеры в дереве настроек, которую необходимо вывести на монитор
	name<>	Имя объекта, которое будет отображаться в левом нижнем углу
	arch_id<>	-
	control<>	0 – только просмотр архива, 1 – также возможно и управление (постановка/снятие с охраны, запись)
	gate_id<>	Идентификатор видеошлюза, через который необходимо получать видео для отображения. Соответствующая камера должна быть добавлена и настроена в данном видеошлюзе – см. Выбор и настройка видеокамер для модуля Видеошлюз
	slave_id<>	Идентификатор компьютера, к которому применяется команда
	stream_id<>	Идентификатор потока по номеру, например <4.1>, или типу, например <stream_client_flag>, <stream_analytics_flag>, <stream_archive_flag>, <stream_alarm_flag>
"ACTIVATE_CAM" – делает активной камеру	cam<>	ID камеры в дереве настроек, которую необходимо сделать активной
"ARCH_FRAME_TIME" – поиск видеоархива по дате и времени	cam<>	-
	date<>	-
	time<>	-

Команда – описание команды	**Параметры**	**Описание параметров**
	mode<>	Может принимать следующие значения: • 0 – видеошлюз, если задан (если не задан, то видеосервер) • 1 – видеосервер • 2 – долговременный архив • 10 + id объекта Внешнее хранилище на панели настройки объекта Монитор (в общем случае 11) – внешнее хранилище
"SETUP" – устанавливает параметры монитора	no_update<>	-
	overlay<>	Включение режима ускорения отображения
	x<>	Координата левого верхнего угла (0 – 100)
	y<>	Координата левого верхнего угла (0 – 100)
	w<>	Размер по горизонтали (0 – 100)
	h<>	Размер по вертикали (0 – 100)
	max_cams<>	Максимально допустимое число камер на мониторе
	min_cams<>	Минимально допустимое число камер на мониторе
	compress<>	-
	panel<>	Показать панель управления (0 – выключена, 1 – включена)
	panel_type<>	-
	s<>	-

Команда – описание команды	Параметры	Описание параметров
	layout<>	-
	gate<>	-
	map_id<>	-
	enable<>	-
	topmost<>	1 – показывать экран поверх всех остальных окон
	type<>	Тип объекта «Монитор»
	allow_move<>	Разрешить перемещение окна
	arch_id<>	Идентификатор архива
	cycle<>	Задержка при автоматическом листании (1 – 20 сек)
	flags<>	Флаги
	name<>	Имя объекта
	overlay<>	Включение режима ускорения отображения (0 – нет ускорения, 1 – ускорение «режим Оверлей», 2 – ускорение «режим DirectDraw»)
	tel_prior<>	Приоритет телеметрии
	gstream_version<>	Если значение не задано, функция автоматического выбора потока отключена. При значении параметра minBPS поток для отображения выбирается автоматически, как описано в разделе Настройка автоматического выбора видеопотока для отображения

Команда – описание команды	**Параметры**	**Описание параметров**
"ACTIVATE" – активирование панели управления монитора	user_id<>	Идентификатор пользователя
	panel_active<>	-
"DEACTIVATE" – де активирование панели управления монитора	-	-
"EXPORT_FRAME" – экспорт кадра в JPG-файл	cam<>	-
	file	-
"KEY_PRESSED" – управление кнопками монитора видеонаблюдения и архива видеозаписей	number<>	-
	cam_id<>	ID камеры, к Окну видеонаблюдения которой требуется применить команду. Если идентификатор не указан, то команда применяется к активному Окну видеонаблюдения (см. Активное Окно видеонаблюдения)

Команда − описание команды	**Параметры**	**Описание параметров**
	key<>	Возможные значения: "ARCH_EDIT_DATE" − изменить дату поиска по архиву; "ARCH_EDIT_TIME" − изменить время поиска по архиву; "ARCH_EDIT_ENTER" − ввод изменений значений в архиве; "ARCH_EDIT_ESCAPE" − отменить редактирование архива; "ARCH_EDIT_BACK"; "ARCH_EDIT_REPLACE"; "WINDOW_ZOOM_IN" − развернуть окно видеонаблюдения; "WINDOW_ZOOM_OUT" − свернуть окно видеонаблюдения; "ZOOM_IN" − приближение изображения; "ZOOM_OUT" − отдаление изображения; "CYCLE_REW" − листание окон видеонаблюдения назад; "CYCLE_FF" − листание окон видеонаблюдения вперед; "LEFT" - сдвиг кадра влево в режиме Zoom; "RIGHT" − сдвиг кадра вправо в режиме Zoom; "UP" − сдвиг кадра вверх в режиме Zoom; "DOWN" − сдвиг кадра вниз в режиме Zoom; "MODE_VIDEO" − режим видеонаблюдения; "MODE_ARCH" − режим воспроизведения архивных видеозаписей; "MODE_ARCH2"- режим воспроизведения архивных видеозаписей 2; "MASK_SHOW" − нанести маску; "MASK_HIDE" − удалить маску; "ARM" − поставить камеру на охрану; "DISARM" − снять камеру с охраны; "REW" − обратная перемотка; "PLAY" − воспроизведение; "PLAY_NONSTOP" − безостановочное воспроизведение; "PLAY_FAST" − ускорить просмотр видеозаписи; "FF" − перемотка вперед; "RECORD" − запись;

Команда – описание команды	**Параметры**	**Описание параметров**
		"RECORD_MIC" – запись с микрофона; "STOP" – остановка; "REC_STOP" – остановка записи; "PAUSE" – пауза; "MIC_ON" – микрофон включен; "MIC_OFF" – микрофон выключен; "PRINT" – вывод кадра на печать. "SELECT_LAYOUT" – управление раскладкой монитора видеонаблюдения; "START_CYCLE_FF" – включение функции автоматического листания окон видеонаблюдения вперед. Период листания задается при настройке интерфейсного объекта **Монитор** (см. Настройка режима отображения окон видеокамер) "STOP_CYCLE" – остановка автоматического листания Окон видеонаблюдения; "SCREEN.N" – выбор раскладки Окон видеонаблюдения. N принимает значения 1, 4, 6, 9, 16, 32 (максимальное значение зависит от количества Окон видеонаблюдения, добавленных в Монитор видеонаблюдения); "EXPORT_DO" – запустить утилиту фонового экспорта AviExport (см. Утилита AviExport); "PROTECT_DO" – открыть окно создания закладки (см. Создание закладок); "PROTECT_VIEW" – открыть список закладок (см. Список закладок)
"START_AVI_EXPORT" – начать экспорт видео *Примечание. См. пример использования в* Примеры с Камерами и Монитором видеонаблюдения	start<>	Время начала
	finish<>	Время окончания
	avi_path<>	Путь к создаваемому файлу
	cam<>	ID камеры
"STOP_AVI_EXPORT" – остановить экспорт видео	monitor<>	Номер монитора

Команда – описание команды	**Параметры**	**Описание параметров**
"START_AVI_SCHEDULE" – начать экспорт закладок	-	-
"STOP_AVI_SCHEDULE" – остановить экспорт закладок	-	-
"CONTROL_TELEMETRY" – Управление телеметрией См. также раздел Управление поворотными устройствами с помощью мыши	cam<>	ID камеры, на которой следует включить или отключить управление телеметрией при помощи мыши
	on<>	0 – отключить управление при помощи мыши 1 – включить управление при помощи мыши
"SET_REC_RESTART" – включить перезапуск записи при входе в архив		
"RESET_REC_RESTART" – отключить перезапуск записи при входе в архив		
"SET_ARCH_ENTER_PAUSE" – включить постановку проигрывания на паузу при входе в архив		
"RESET_ARCH_ENTER_PAUSE" – отключить постановку проигрывания на паузу при входе в архив		
"DISABLE_TELEMETRY" – отключить возможность управления телеметрией из Монитора видеонаблюдения		

Команда – описание команды	**Параметры**	**Описание параметров**
"ENABLE_TELEMETRY" – включить возможность управления телеметрией из Монитора видеонаблюдения		
"INCREASE_VIEW" – увеличить размер окна камеры в Мониторе видеонаблюдения	cam<>	ID камеры
"DECREASE_VIEW" – уменьшить размер окна камеры в Мониторе видеонаблюдения	cam<>	ID камеры
"SHOW_LAYOUT" – отобразить раскладку с указанным идентификатором	layout_id <>	Идентификатор раскладки в базе данных
"GO_LIVE" – переключить все камеры на мониторе в режим живого виде	-	-
"GO_ARCH" – переключить все камеры в мониторе в режим просмотра архива	arch_time <>	Необязательный параметр. Задает время позиционирования в архиве в формате ДД-ММ-ГГ ЧЧ:ММ:СС. По умолчанию архив позиционируется на последнюю запись
"SAVE_AS" – экспортировать выбранный фрагмент архива	-	-
"PLACE_CAM_IN_LAYOUT_CELL" – добавить камеру на Монитор в заданную ячейку заданной раскладки	cam<>	ID камеры в дереве объектов, которую необходимо вывести на монитор. Если значение параметра некорректно, например, 0 или -1, то соответствующая ячейка будет скрыта
	layout_name<>	ID или название раскладки, на которую необходимо добавить камеру

Команда – описание команды	**Параметры**	**Описание параметров**
	cell<>	Номер ячейки на раскладке, в которую необходимо добавить камеру. Ячейки нумеруются сверху вниз и слева направо, начиная с верхнего левого угла раскладки. ***Внимание!*** *Нумерация ячеек начинается с 0.* Если в ячейку уже добавлена какая-либо другая камера, она будет заменена
"SET_TITLES" – показывать титры поверх видеоизображения в любом режиме отображения. Такие титры не записываются в архив и отображаются до применения команды CLEAR_TITLES либо перезагрузки Монитора	cam<>	ID камеры, к Окну видеонаблюдения которой требуется применить команду
	titles<>	Текст титров, который необходимо выводить. Для переноса строки используется '\r'
	title_id<>	ID титрователя
"CLEAR_TITLES" – выключить показ титров, созданных с помощью команды SET_TITLE	cam<>	ID камеры, к Окну видеонаблюдения которой требуется применить команду
	title_id<>	ID титрователя

Свойства объекта **MONITOR** показаны в таблице:

Свойства объекта MONITOR	**Описание свойств объекта**
ID<>	Идентификатор объекта
PARENT_ID<>	Идентификатор родительского объекта

10.4 MACRO Макрокоманда

Объект **MACRO** соответствует системному объекту **Макрокоманда**.

От объекта **MACRO** поступают события, представленные в таблице ниже. Запуск процедур происходит при возникновении соответствующего события.

События	**Описание событий**	**Параметры**	**Описание параметров**
RUN	Выполнено действие	src_sender<>	Имя компьютера, на котором была запущена макрокоманда. *Примечание. Значение данного параметра будет отображаться в Протоколе событий в столбце **Доп. инфо** в реальном времени. При перезапуске ПК Интеллект и загрузке записей протокола событий из БД данная информация не отображается в интерфейсе, но остается в базе данных*
		user_id<>	Идентификатор пользователя, выполнившего макрокоманду. *Примечание. Значение данного параметра вместе с именем пользователя будет отображаться в Протоколе событий в столбце **Доп. инфо** в реальном времени. При перезапуске ПК Интеллект и загрузке записей протокола событий из БД данная информация не отображается в интерфейсе, но остается в базе данных*

Список команд и параметров для объекта **MACRO** представлен в таблице:

Свойства объекта **MACRO** показаны в таблице:

Свойства объекта MACRO	**Описание свойств объекта**
ID<>	Идентификатор объекта
PARENT_ID<>	Идентификатор родительского объекта

Объект **MACRO** может находиться в состояниях, описанных в таблице:

Состояние объекта MACRO	**Описание состояния объекта**
"NORM"	Норма

10.5 SLAVE Компьютер

Объект **SLAVE** соответствует системному объекту **Компьютер**.

От объекта **SLAVE** поступают события, представленные в таблице ниже. Запуск процедур происходит при возникновении соответствующего события.

События	**Описание событий**	**Комментарий**
CONNECTED	Подключение	Событие генерируется, когда какой-либо Клиент подключился к Серверу
DISCONNECTED	Отключение	Событие генерируется, когда какой-либо Клиент отключился от Сервера
KEY_IGNORED_HW	Ключ отвергнут (несоответствие кодов плат)	Событие генерируется в случае, если коды плат (либо HID) в ключе не соответствуют текущим у компьютера
KEY_IGNORED_SW	Ключ отвергнут (превышено ограничение)	Событие генерируется при наличии софтверных ограничений. Например, когда ключ подходит, но количество созданных в дереве оборудования объектов больше, чем указано в ключе
KEY_UPDATED	Ключ обновлен	
PROTOCOL_RCVD	Протокол получен	
REBUILD_IN_START	Начало переиндексации архива	
REBUILD_IN_STOP	Окончание переиндексации архива	
REGISTER_ATTEMPT	Попытка несанкционированного доступа	

События	**Описание событий**	**Комментарий**
REGISTER_ERROR	Превышен лимит попыток доступа	Событие генерируется, когда пользователь много раз предпринимал неудачные попытки входа в систему. После события возникает некоторый таймаут, когда данный пользователь не сможет сделать попытку входа. Количество попыток входа и таймаут можно изменить через реестр
REGISTER_USER	Регистрация пользователя	Данное событие происходит при попытке пользователя войти в систему (при вводе логина и пароля)
DISC_EXIST	Диск для записи архива присутствует	
NO_DISC	Диск для записи архива отсутствует	
KEY_IGNORED_FR	Ключ отвергнут	Событие генерируется в случае, если ключевой файл не удалось записать на диск
SHUTDOWN	Завершение работы	
DISC_MOUNT	Диск подключен (монтирован)	
DISC_UNMOUNT	Диск отсоединен (размонтирован)	

События	**Описание событий**	**Комментарий**
ARCHIVE_DEPTH	Глубина архива	Событие генерируется в полночь и содержит информацию о глубине архива по всем дискам в часах (параметр depth<>). Для вызова события вручную используется реакция GET_DEPTH. При отображении события в Протоколе событий в поле **Дополнительная информация** указывается глубина архива в формате Дни:Часы. Также данная информация содержится в параметре события param0<>. Глубина архива рассчитывается как разница между датами создания самого старого файла архива и самого нового файла архива (на диске или по камере)
FORCED_OFF	Принудительная выгрузка	Событие генерируется перед принудительной выгрузкой ПК *Интеллект*, например, в случае, если извлечен ключ защиты Guardant. Выгрузка производится после повлекшего ее действия (например, извлечения ключа Guardant) через интервал времени, задаваемый ключом реестра UnloadDelay – см. Справочник ключей реестра
DEACTIVATE_ALL_DISP	Скрыть все экраны	Событие позволяет скрыть все экраны на указанном в параметре slave<> компьютере. Если в событии присутствует параметр except<>, то скрываются все экраны, кроме экрана с указанным в данном параметре идентификатором
LIC_EXPIRATION	Действие лицензии заканчивается через	По умолчанию не генерируется. Для включения необходимо установить ключ NotifyExpireLic = 1 (см. Справочник ключей реестра). В параметре days<> указывается количество дней до окончания лицензии (может быть дробным числом). Событие генерируется в момент загрузки ПК *Интеллект* и после смены дня

События	**Описание событий**	**Комментарий**
DATABASE_ERROR	Потеряна связь с базой данных	Событие генерируется в случае разрыва связи с SQL Server при первом обращении к нему после разрыва
SCRIPT_ERROR	Выполнение скрипта завершилось ошибкой	По умолчанию событие не генерируется, так как не добавлено в файл внешних настроек intellect.ddi. Чтобы событие SCRIPT_ERROR генерировалось и записывалось в таблицу PROTOCOL, его необходимо добавить в таблицу intellect.ddi (см. Редактирование файла внешних настроек intellect.ddi с помощью утилиты ddi.exe)

Список команд и параметров для объекта **SLAVE** представлен в таблице:

Команда – описание команды	**Параметры**	**Описание параметров**
"**SETUP**" – установить параметры для компьютера	display_id<>	Идентификатор экрана
	drives<>	Диски для записи видеоархива
	drives_a<>	Диски для записи аудиоинформации
	flags<>	Флаги
	arch_days<>	Размер архива событий
	connection<>	Соединение
	disable_protocol<>	Отключить протоколирование
	ip_address<>	IP адрес устройства
	is_backup<>	Архивация
	is_load<>	Загружен

Команда – описание команды	**Параметры**	**Описание параметров**
	local_protocol<>	Локальный протокол
	modem<>	Модемное соединение
	name<>	Имя объекта
	password<>	Пароль
	sync_time<>	Синхронизировать время
	username<>	Имя пользователя
"BACKUP" – сделать резервную копию БД	-	-
"CONNECT_ONE" – подключиться к компьютеру. Подключает соответствующий компьютер. Следует избегать использования этой реакции вручную	-	-
"CONNECT_OTHER" – подключиться к ядрам. Подключает компьютер к другим ядрам из конфигурации. Следует избегать использования этой реакции вручную	-	-
"DISCONNECT_ONE" – отключиться от компьютера. Отключает соответствующий компьютер. В случае отключения ядро может автоматически подключиться. Следует избегать использования этой реакции вручную	-	-
"SYNC_PROTOCOL" – запустить утилиту синхронизации протокола SyncProtocol.exe. Если синхронизация настроена, происходит слияние протокола	-	-

Команда – описание команды	**Параметры**	**Описание параметров**
"SYNC_TIME" – синхронизовать время. Для выполнения данной реакции необходимо, чтобы в разделе реестра HKEY_LOCAL_MACHINE\SOFTWARE\ITV\INTELLECT\ (HKEY_LOCAL_MACHINE\Software\Wow6432Node\ITV\INTELLECT для 64-битной системы) был создан параметр SyncTime со значением 1 на той системе, которой адресована реакция	-	-
"CREATE_PROCESS" – запустить процесс	command_line<>	Командная строка. Команды командной строки Windows, записанные без символов переноса строки через разделители |, & или &&
"SEND_MY_CONFIG" – разослать конфигурацию. Рассылает свою конфигурацию другим компьютерам. То же, что “SPREAD_CONFIG”	-	-
"MOVE_CONFIG" – переместить конфигурацию. Перемещает конфигурацию, созданную в дереве объектов на основе компьютера-Поставщика, на компьютер-Получатель	from<>	Поставщик
	to<>	Получатель
"SPREAD_CONFIG" – распространить конфигурацию, то же, что “SEND_MY_CONFIG”	-	-
"GET_DEPTH" – получить глубину архива. В ответ на реакцию в системе формируется событие ARCHIVE_DEPTH (см. таблицу выше). Отсутствие одного или обоих параметров означает запрос глубины по записям для всех возможных значений параметра	cam<>	Идентификатор камеры, для которой запрашивается глубина архива

Команда – описание команды	**Параметры**	**Описание параметров**
	drive<>	Диск или сетевой путь, по которому запрашивается глубина архива. Название диска задается формате "<буква диска>:\\", например drive<D:\\> *Примечание. Символ "\" – экранируемый.* Сетевой путь задается в формате UNC
"ACTIVATE_DISPLAY" – сменить экран. Команда позволяет отобразить на мониторе (мониторах) компьютера Экран с заданным идентификатором	display_id<>	Идентификатор соответствующего объекта **Экран**. Если в параметре передано пустое значение, при выполнении данной команды скрываются все экраны

Свойства объекта **SLAVE** показаны в таблице:

Свойства объекта SLAVE	**Описание свойств объекта**
ID<>	Идентификатор объекта
PARENT_ID<>	Идентификатор родительского объекта
USER_ID<>	Идентификатор пользователя

10.6 DISPLAY Экран

Объект **DISPLAY** соответствует системному объекту **Экран**.

От объекта **DISPLAY** поступают события, представленные в таблице ниже. Запуск процедур происходит при возникновении соответствующего события.

Событие	**Описание события**
ACTIVATE	Активация экрана

DEACTIVATE	Деактивация экрана
ACTIVATED	Активация экрана на удаленном компьютере. В параметре param0<> передается имя компьютера

Список команд и параметров для объекта **DISPLAY** представлен в таблице:

Команда – описание команды	**Параметры**	**Описание параметров**
"ACTIVATE" – показать экран	macro_slave_id<>	Имя компьютера, на котором должен быть показан экран
"DEACTIVATE" – скрыть экран	macro_slave_id<>	Имя компьютера, на котором должен быть скрыт экран

Примечание.

Если параметр «macro_slave_id» не установлен, то команда будет выполнена для всех компьютеров в системе.

Свойства объекта **DISPLAY** показаны в таблице:

Свойства объекта DISPLAY	**Описание свойств объекта**
flags	Флаги
id	Идентификатор объекта
name	Имя объекта
parent_id	Идентификатор родительского объекта

10.7 PLAYER Аудиопроигрыватель

Объект **PLAYER** соответствует системному объекту **Аудиопроигрыватель**.

Список команд и параметров для объекта **PLAYER** представлен в таблице:

Команда – описание команды	**Параметры**	**Описание параметров**
"PLAY_WAV" – проигрывает звуковой файл	file<>	Полный путь к звуковому файлу в формате .wav (с указанием имени проигрываемого файла, например: C:\Program Files (x86)\Intellect\Wav\cam_alarm_1.wav)
	from_macro<ID>	Флаг реакции проигрывания звукового файла, отправленной с макрокоманды (с указанием ID существующей макрокоманды, ID > 0) *Примечание. Параметр не является обязательным, если канал голосового оповещения настроен в интерфейсе объекта **Аудиопроигрыватель** (см. Настройка голосового оповещения с помощью объекта Аудиопроигрыватель)*
"SETUP" – настройка параметров аудиопроигрывателя	board<>	Звуковое устройство проигрывателя архива
	flags<>	Флаги
	h<>	Высота диалога настройки (0 – 100)
	name<>	Имя объекта
	voice<>	Звуковое оповещение
	voice_board<>	Звуковое устройство оповещения
	w<>	Ширина диалога настройки (0 – 100)
	x<>	Левый верхний угол диалога настройки (0 – 100)
	y<>	Левый верхний угол диалога настройки (0 – 100)

Команда – описание команды	Параметры	Описание параметров
"STOP_WAV" – завершение проигрывания файла	-	-

Свойства объекта **PLAYER** показаны в таблице:

Свойства объекта PLAYER	Описание свойств объекта
ID<>	Идентификатор объекта
PARENT_ID<>	Идентификатор родительского объекта

10.8 CORE Ядро

Объект **CORE** – это ядро системы, глобальный статический объект, реализующий методы, используемые для контроля состояния и управления системными объектами программного комплекса *Интеллект*. Более широкие возможности для работы с объектом CORE предоставляются при использовании скриптов на языке программирования JScript – см. документ Объект Скрипт. Программирование с использованием языка JScript

От объекта CORE поступают события, представленные в таблице:

Событие	Описание события

DO_REACT	Событие инициирует реакцию того или иного объекта в системе. В параметре action данного события передается описание действия, которое требуется выполнить. Примеры значений параметра action: SET_MARKRECT – посылается при обнаружении лица на видеоизображении; DEL_MARKRECT – посылается при исчезновении лица с видеоизображения. Также могут присутствовать другие параметры события, которые можно отследить при помощи Отладочного окна (см. Отладочное окно). Например, если значение параметра action равно SET_MARKRECT, то в параметре param5_val передается номер камеры, на видеоизображении с которой обнаружено лицо. Об этом говорит название параметра, передаваемое в параметре param5_name. Для значения DEL_MARKRECT номер камеры передается в параметре param0_val
SLAVE_CHANGED	Событие генерируется при срабатывании Сервиса отказоустойчивости (Failover). Содержит следующие параметры: • old_slave_id – идентификатор объекта **Компьютер**, с которого переносятся камеры • new_slave_id – идентификатор объекта **Компьютер**, на который переносятся камеры • CAM<n1, n2, ... > – где n1, n2 и т.д. являются идентификаторами камер, перенесенных под другой родительский объект **Компьютер**. Например, CAM<4,6,7> – перенос камер с идентификаторами 4, 6, 7
CREATE_OBJECT	Событие инициирует создание объекта. Параметры: • objtype<> – тип объекта, например, objtype<PERSON> – создание пользователя • parent_id<> – идентификатор родительского объекта • service_photo<> – при создании пользователя в данном параметре передается кодированное в base64 бинарное изображение – фотография пользователя. Параметр необходим, чтобы при создании пользователя в Бюро пропусков ему можно было сразу назначить фотографию

10.9 MAP Карта

Объект **MAP** соответствует системному объекту **Карта**.

От объекта **MAP** поступают события, представленные в таблице ниже. Запуск процедуры происходит при возникновении соответствующего события.

Событие	**Описание события**
LAYER_ACTIVATED	Активация слоя. Данное событие поступает при переходе на слой карты. В параметре obj_id<> содержится идентификатор активированного слоя
ACTIVATE_OBJECT	Активация объекта. Событие поступает при выборе (активации кликом мыши) объекта на карте. Параметры: 1. obj_type<> – тип объекта 2. user_id<> – идентификатор пользователя 3. module<> – имя модуля, для Карты – map.run 4. date<> – дата возникновения события 5. time<> – время возникновения события 6. slave_id<> – сетевое имя компьютера 7. obj_id<> – идентификатор объекта 8. layer<> – идентификатор слоя Карты 9. fraction<> – миллисекунда, в которую событие возникло 10. owner<> – пользователь, активировавший объект 11. type_of_display<> – тип отображения объекта, возможные значения: a. IMAGE – изображение b. IMAGE_AND_INDICATOR – изображение и индикатор c. TEXT – текст d. LINE – линия e. POLYGON – многоугольник f. ELIPSIS – эллипс g. TITLE – название объекта
OBJDBLCLK	Событие поступает при двойном клике по объекту на Карте. Содержит те же параметры, что ACTIVATE_OBJECT

Список команд и параметров для объекта **MAP** представлен в таблице:

Команда – описание команды	**Параметры**	**Описание параметров**
"SET_TOPMOST" – поверх всех окон	-	-

"SET_NOTOPMOST" – отмена поверх всех окон	-	-
"HIDE_OBJECT" – скрыть/ показать значки объектов на карте	objtype<>	Тип объекта. Может быть пустым. Если тип объекта не задан, скрываются/ отображаются объекты всех типов
	objid<>	Идентификатор объекта. Может быть пустым. Если идентификатор объекта не задан, скрываются/отображаются все объекты заданного типа
	hide<>	0 – объекты отображаются на карте 1 – объекты не отображаются на карте
"SET_OBJECT_GEOMETRY" – задать положение объекта на карте	objtype<>	Тип объекта
	objid<>	Идентификатор объекта
	x<>	Новая координата верхнего левого угла значка объекта на слое карты в пикселях по оси X
	y<>	Новая координата верхнего левого угла значка объекта на слое карты в пикселях по оси Y
	exclude_children<>	По умолчанию при использовании реакции SET_OBJECT_GEOMETRY при перемещении значков объектов перемещаются и названия этих объектов (дочерние объекты). Если передать в реакции параметр exclude_children<1>, то объект перемещается отдельно от дочерних, то есть без названия
"INSCRIBE" – вписать в окно	-	-
"SHOW_MINIMAP" – показать миникарту	x<>	Координата верхнего левого угла миникарты по оси X в пикселях

	y<>	Координата верхнего левого угла миникарты по оси Y в пикселях
	w<>	Ширина миникарты в пикселях
	h<>	Высота миникарты в пикселях
	monitor<>	Идентификатор монитора
	slave_id<>	Сетевое имя компьютера
"SET_ZOOM" – изменить масштаб карты	zoom<>	Задаваемый масштаб карты
"ACTIVATE_OBJECT" – активировать объект на карте	obj_type<>	Тип объекта
	obj_id<>	Идентификатор объекта
	layer<>	Идентификатор слоя карты. Если параметр указан, то скрипт будет работать на указанном слое, если не указан, то на текущем слое
"DRAW_ARROW" – нарисовать трек перемещения между объектами	first_obj_type<>	Тип объекта, от которого будет строиться трек
	first_obj_id<>	Идентификатор объекта, от которого будет строиться трек
	second_obj_type<>	Тип объекта, к которому будет строиться трек
	second_obj_id<>	Идентификатор объекта, к которому будет строиться трек
	obj_id<>	Идентификатор создаваемого трека

	title_text<>	Текст, который будет отображаться рядом с треком. Для переноса строки используется \n Дополнительные необязательные параметры: • title_text_align – смещение текста. 0 – текст отображается по центру, 1 – в начале трека, 2 – в конце трека • title_text_size – размер текста • title_shift_by_y – смещение текста вниз (может понадобиться при использовании title_text_size). По умолчанию 40 пикселей • title_text_color – цвет текста в формате Decimal • title_text_font – шрифт текста Пример: title_text<Объект движется\nк выходу>,title_text_align<1>,title_shift_by_y<60>,title_text_color<16711935>,title_text_size<18>,title_text_font<Algerian>
"ERASE_ARROW" – удалить трек перемещения между объектами	obj_id<>	Идентификатор трека, который надо удалить. Если не указывать параметр, то будут удалены все треки

Особенности реализации команды **DRAW_ARROW**:

1. В результате запуска команды трек будет отображаться на каждом слое **Карты** в виде стрелок  между заданными объектами.
2. Если объекты расположены на одном слое, то стрелка рисуется напрямую между указанными объектами. Если объекты на разных слоях, то стрелка рисуется по самому короткому пути.
3. Можно ограничить глубину поиска взаимосвязей по слоям для построения трека ключом DrowArrowSearchDepth, см. Справочник ключей реестра.
4. Если
   a. невозможно построить трек,
   b. одного из объектов не существует,
   c. можно построить трек, но невозможно отобразить стрелки,

   то на стартовом объекте будет отображаться  , на конечном  .

10.10 OLXA_LINE Микрофон

Объект **OLXA_LINE** соответствует системному объекту **Микрофон**.

От объекта **OLXA_LINE** поступают события, представленные в таблице ниже. Запуск процедуры происходит при возникновении соответствующего события.

Событие	**Описание события**
ACCU_START	Включение акустопуска
ACCU_STOP	Выключение акустопуска
ARM	Запись включена
DISARM	Запись выключена
INCOMING_NUMBER	Входящий телефонный номер
OUTGOING_NUMBER	Исходящий телефонный номер
REC	Начало записи
REC_STOP	Конец записи
RESET	Подключение микрофона

Список команд и параметров для объекта **OLXA_LINE** представлен в таблице:

Команда – описание команды	**Параметры**	**Описание параметров**
"ARM" – включить микрофон на запись	-	-
"DISARM" – выключить запись с микрофона	-	-
"SETUP" – настройка параметров микрофона	type<>	Тип линии

Команда – описание команды	**Параметры**	**Описание параметров**
	accu_start<>	Порог срабатывания детектора звука
	accu_stop<>	Время удержания сработки детектора
	amp<>	Усиление
	aru<>	Автоматическая регулировка усиления
	aru_dyn<>	Уровень АРУ
	aru_time<>	Время срабатывания АРУ
	chan<>	Номер звукового канала микрофона
	compression<>	Тип компрессии
	flags<>	Флаги
	name<>	Имя объекта
	rec<>	Начало записи

Свойства объекта **OLXA_LINE** показаны в таблице:

Свойства объекта OLXA_LINE	**Описание свойств объекта**
ID<>	Идентификатор объекта
PARENT_ID<>	Идентификатор родительского объекта

Объект **OLXA_LINE** может находиться в состояниях, описанных в таблице:

Состояние объекта OLXA_LINE	Описание состояния объекта
"BLUE"	Микрофон снят с охраны
"GREEN"	Нет сигнала от микрофона
"YELLOW"	Микрофон поставлен на охрану
"RED"	Начало записи

10.11 TELEMETRY Поворотное устройство

Объект **TELEMETRY** соответствует системному объекту **Поворотное устройство**.

От объекта **TELEMETRY** поступают события, представленные в таблице ниже. Запуск процедур происходит при возникновении соответствующего события.

Событие	Описание события	Комментарий
LOCKED	Заблокировано	Событие поступает после команды LOCK (см. таблицу ниже)
UNLOCKED	Разблокировано	Событие поступает после команды UNLOCK (см. таблицу ниже)

Список команд и параметров для объекта **TELEMETRY** представлен в таблице:

Команда – описание команды	Параметры	Описание параметров
"AUTOFOCUS_ON" – включить функцию автонаведения (автофокус)	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"AUTOPAN_END_P" – задать конечную точку автоповорота	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"AUTOPAN_START" – начать автоповорот	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)

Команда – описание команды	**Параметры**	**Описание параметров**
"AUTOPAN_START_P" – задать стартовую точку автоповорота	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"AUTOPAN_STOP" – окончить автоповорот	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"CLEAR_PRESET" – очистить выбранный пресет	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
	preset<>	Пресет
"D2OFF" – отключить дополнительные динамические настройки для поворотных видеокамер Panasonic, предназначенные для улучшения качества аналогового видеосигнала	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"D2ON" – включить дополнительные динамические настройки для поворотных видеокамер Panasonic, предназначенные для улучшения качества аналогового видеосигнала	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"DOWN" – повернуть объектив видеокамеры вниз	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"FOCUS_IN" – увеличить изображение	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"FOCUS_OUT" – уменьшить изображение	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"FOCUS_STOP" – остановить увеличение/уменьшение изображения	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"GO_PRESET" – повернуть видеокамеру в положение, заданное на пресете	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
	preset<>	Пресет

Команда – описание команды	**Параметры**	**Описание параметров**
"HOME" – повернуть видеокамеру в исходную (домашнюю) позицию	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"IRIS_CLOSE" – закрыть диафрагму	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"IRIS_OPEN" – открыть диафрагму	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"IRIS_STOP" – остановить диафрагму	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"LEFT" – повернуть объектив видеокамеры влево	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"LEFT_DOWN" – повернуть объектив видеокамеры влево и вниз	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"LEFT_UP" – повернуть объектив видеокамеры влево и вверх	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"PATROL_LEARN" – начать процедуру программирования патрулирования, выполняемую путем записи поведения видеокамеры	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
	point<>	Номер точки
	preset<>	Номер пресета (тура)
	dwell<>	Время нахождения в точке в секундах
	speed<>	Скорость перемещения в точку
	flush_tour<>	1 – записать тур 0 – не записывать тур
"PATROL_PLAY" – начать патрулирование	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)

Команда – описание команды	**Параметры**	**Описание параметров**
"PATROL_STOP" – закончить патрулирование	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"RIGHT" – повернуть объектив видеокамеры вправо	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"RIGHT_DOWN" – повернуть объектив видеокамеры вправо и вниз	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"RIGHT_UP" – повернуть объектив видеокамеры вправо и вверх	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"SET_PRESET" – записать текущее положение видеокамеры в выбранный пресет	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
	preset<>	Пресет
"STOP" – завершить поворот объектива видеокамеры	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"UP" – повернуть объектив видеокамеры вверх	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий)
"SETUP" – настроить поворотное устройство	address<>	Адрес устройства
	cam<>	Идентификатор камеры для управления
	flags<>	Флаг работы объекта (0 – включен, 1 – отключен)
	name<>	Имя объекта поворотного устройства
	speed<>	Скорость

Команда – описание команды	**Параметры**	**Описание параметров**
"SEND_BUFFER" – отправить команду в шестнадцатеричном формате в COM-порт	buffer<>	Команда в шестнадцатеричном формате
	parent_id<>	Номер родительского объекта **Контроллер телеметрии.** Обязательный параметр
	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий). Значение параметра должно быть больше 0
"LOCK" – заблокировать. Перевод телеметрии в состояние LOCKED на заданное время	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий). Значение параметра должно быть больше 0. На время блокировки запрещается выполнение команд управления с более низким приоритетом, чем указанный
	duration<>	Длительность наложения блокировки. Если параметр не указан, блокировка действует до выполнения команды UNLOCK
"UNLOCK" – разблокировать. Перевод телеметрии в состояние UNLOCKED	-	-
"AUTOFOCUS_OFF" – выключить функцию автонаведения (автофокус)	tel_prior<>	Приоритет (1 – низкий, 2 – средний, 3 – высокий). ⚠️ Для использования этой команды её необходимо добавить на вкладку **Реакции** для объекта **TELEMETRY** в ddi.exe (см. Закладка Реакции)

Свойства объекта **TELEMETRY** показаны в таблице:

Свойства объекта TELEMETRY	**Описание свойств объекта**
ID<>	Идентификатор объекта поворотного устройства
PARENT_ID<>	Идентификатор родительского объекта

Объект **TELEMETRY** может находиться в состояниях, описанных в таблице:

Состояние объекта TELEMETRY	**Описание состояния объекта**
LOCKED – Заблокировано	Управление телеметрией заблокировано с некоторым приоритетом. Запрещено управление телеметрией с приоритетом ниже указанного при блокировке (см. таблицу выше)
UNLOCKED – Разблокировано	Разрешено управление телеметрией с любым приоритетом

10.12 TELEMETRY_EXT Пульт управления

Объект **TELEMETRY_EXT** соответствует системному объекту **Пульт управления**.

От объекта **TELEMETRY_EXT** поступают события, представленные в таблице ниже. Запуск процедур происходит при возникновении соответствующего события.

Событие	**Описание события**	**Параметр**	**Описание параметра**	**Диапазон значений**
KEY_PRESSED	Нажата клавиша	param 0<>	Код нажатой клавиши	См. Руководство по установке и настройке компонентов охранной системы
		device <>	Устройство, на котором нажата клавиша	0 – Основная клавиатура *AXIS T8312* 1 – Клавиатура *AXIS T8313*
KEY_RELEASED	Отпущена клавиша	param 0<>	Код отпущенной клавиши	0..21 для *AXIS T8312*. Для *BOSCH KBD-Digital, BOSCH KBD-Universal* и *Panasonic WV-CU950* см. Руководство по установке и настройке компонентов охранной системы
		device <>	Устройство, на котором отпущена клавиша	0 – Основная клавиатура *AXIS T8312* 1 – Поворотный переключатель *AXIS T8313*

Событие	**Описание события**	**Параметр**	**Описание параметра**	**Диапазон значений**
MOVED	Изменено положение	param 0<>	Значение смещения	Для колеса поворотного переключателя JogDial -1.. 1; для колеса покадровой прокрутки Shuttle -7..7 Для пульта *Panasonic WV-CU950* JogDial -1.. 1; Shuttle -6..6
		device <>	Тип использованного механизма управления *AXIS T8313*	0 – колесо поворотного переключателя 1 – колесо покадровой прокрутки

Список команд и параметров для объекта **TELEMETRY_EXT** представлен в таблице:

Команда – описание команды	**Параметры**	**Описание параметров**
"DRAW_FIGURE" – нарисовать фигуру на дисплее пульта телеметрии *BOSCH KBD-Digital или BOSCH KBD-Universal*	display<>	0x00 – основной дисплей, 0x01 – статусный дисплей
	x1<>	Начальная координата по оси X (от 0 до 127 для основного дисплея, от 0 до 121 для статусного)
	y1<>	Начальная координата по оси Y (от 0 до 239 для основного дисплея, от 0 до 31 для статусного)
	x2<>	Конечная координата по оси X (от 0 до 127 для основного дисплея, от 0 до 121 для статусного)
	y2<>	Конечная координата по оси Y (от 0 до 239 для основного дисплея, от 0 до 31 для статусного)
	is_fill<>	0 – не закрашивать фигуру, 1 – закрашивать фигуру

Команда – описание команды	**Параметры**	**Описание параметров**
	is_set_pixels<>	0 – стереть фигуру с дисплея, 1 – нарисовать фигуру
	figure<>	0 – линия, 1 – прямоугольник
"PRINT_TEXT" – напечатать текст на дисплее пульта телеметрии *BOSCH KBD-Digital или BOSCH KBD-Universal*	display<>	0x00 – основной дисплей, 0x01 – статусный дисплей
	x<>	Координата по оси X (от 0 до 127 для основного дисплея, от 0 до 121 для статусного)
	y<>	Координата по оси Y (от 0 до 239 для основного дисплея, от 0 до 31 для статусного)
	charset<>	Кодировка: 0 – Латинская 1 – Кириллическая 2 – Центральноевропейская
	style<>	Стиль: 0 – Обычный 1 – Полужирный
	text<>	Текстовое сообщение
"PRINT_TEXT" – напечатать текст на дисплее пульта телеметрии *Panasonic WV-CU950*	y<>	0 – вывести текст на первую строку 1 – вывести текст на вторую строку
	text<>	Выводимый текст строки, максимум 20 символов

Команда – описание команды	**Параметры**	**Описание параметров**
	flickering<>	Строка из шести символов, определяющая параметры мигания текста: d1 d2 d3 d4 d5 d6 d1 определяет период мигания: 0 – мигание отключено 1 – период 0.25 сек, символ заменяется белым пробелом 2 – период 0.5 сек, символ заменяется белым пробелом 3 – период 0.75 сек, символ заменяется белым пробелом 4 – период 1 сек, символ заменяется белым пробелом 5 – период 0.25 сек, символ заменяется темным пробелом 6 – период 0.5 сек, символ заменяется темным пробелом 7 – период 0.75 сек, символ заменяется темным пробелом 8 – период 1 сек, символ заменяется темным пробелом d2: 1 – мигают символы с 1 по 4, 0 – данные символы не мигают d3: 1 – мигают символы с 5 по 8, 0 – данные символы не мигают d4: 1 – мигают символы с 9 по 12, 0 – данные символы не мигают d5: 1 – мигают символы с 13 по 16, 0 – данные символы не мигают d6: 1 – мигают символы с 17 по 20, 0 – данные символы не мигают
"CLEAR_DISPLAY" – очистить дисплей пульта телеметрии *BOSCH KBD-Digital или BOSCH KBD-Universal*. Для пульта телеметрии *Panasonic WV-CU950* реакция без параметров	display<>	0x00 – основной дисплей, 0x01 – статусный дисплей

Команда – описание команды	**Параметры**	**Описание параметров**
"RELE_ON" – включить лампочку на клавиатуре *AXIS T8312* или пульте *Panasonic WV-CU950*	rele<>	Код клавиши с лампочкой, 12..16 для *AXIS T8312*. Для Panasonic WV-CU950 см. Руководство по установке и настройке компонентов охранной системы, раздел Особенности настройки и работы с пультом управления телеметрией Panasonic WV-CU950
"RELE_OFF" – выключить лампочку на клавиатуре *AXIS T8312* или пульте *Panasonic WV-CU950*	rele<>	Код клавиши с лампочкой, 12..16
"RESET" – физическая перезагрузка пульта *Panasonic WV-CU950*	type<>	0 – немедленная перезагрузка 1 – перезагрузка по истечении 100мсек 2 – перезагрузка по истечении 200мсек 3 – перезагрузка по истечении 500мсек 4 – перезагрузка по истечении 1сек

Команда – описание команды	**Параметры**	**Описание параметров**
"SET_ALARM" – задает вид тревожного сигнала пульта *Panasonic WV-CU950*	audio_alarm<>	0 – звук отключен 1 – простой однократный сигнал тревоги 2 – простой двукратный сигнал тревоги 3 – простой троекратный сигнал тревоги 4 – однократный сигнал тревоги длительностью 0.1 сек 5 – однократный сигнал тревоги длительностью 0.2 сек 6 – однократный сигнал тревоги длительностью 0.3 сек 7 – однократный сигнал тревоги длительностью 1 сек 8 – простое однократное звучание 9 – простое двукратное звучание A – простое троекратное звучание B – однократный сигнал длительностью 0.1 сек C – однократный сигнал длительностью 0.2 сек D – однократный сигнал длительностью 0.3 сек E – однократный сигнал длительностью 1 сек F – сигнал тревоги

10.13 JOYSTICK Устройство управления

Объект **JOYSTICK** соответствует объекту **Устройство управления**.

От объекта **JOYSTICK** поступают события, представленные в таблице. Запуск процедур происходит при возникновении соответствующего события.

События	**Параметр**	**Описание параметра**
KEY_PRESSED – Нажата клавиша	button<>	Код клавиши
	cam<>	Идентификатор камеры
	param0<>	Название УРМ, к которому подключено устройство

10.14 TIME_ZONE Временная зона

Объект **TIME_ZONE** соответствует системному объекту **Временная зона**.

От объекта **TIME_ZONE** поступают события, представленные в таблице ниже. Запуск процедур происходит при возникновении соответствующего события.

События	**Описание событий**
ACTIVATE	Начало
DEACTIVATE	Конец

Список команд и параметров для объекта **TIME_ZONE** представлен в таблице:

Свойства объекта **TIME_ZONE** показаны в таблице:

Свойства объекта TIME_ZONE	**Описание свойств объекта**
ID<>	Идентификатор объекта
PARENT_ID<>	Идентификатор родительского объекта

Объект **TIME_ZONE** может находиться в состояниях, описанных в таблице:

Состояние объекта TIME_ZONE	**Описание состояния объекта**
"ACTIVE"	Активный

Состояние объекта TIME_ZONE	**Описание состояния объекта**
"INACTIVE"	Неактивный

10.15 ARCH Долговременный архив

Объект ARCH соответствует системному объекту **Долговременный архив.**

От объекта ARCH поступают события, представленные в таблице. Запуск процедур происходит при возникновении соответствующего события.

События	**Описание событий**	**Комментарий**
ACTIVE	Долговременный архив активен	Событие генерируется, когда список камер, видео с которых архивируется, соответствует списку конфигурации Долговременного архива
INACTIVE	Долговременный архив неактивен	Событие генерируется, когда не производится архивация через Долговременный архив
ACTIVE_PART	Частичная работа Долговременного архива	Событие генерируется, когда включена архивация видео не от всех камер, указанных в списке Долговременного архива

10.16 FAILOVER Сервис отказоустойчивости

Объект **FAILOVER** соответствует системному объекту **Сервис отказоустойчивости**.

От объекта **FAILOVER** поступают события, представленные в таблице ниже. Запуск процедур происходит при возникновении соответствующего события.

События	**Описание событий**
START	Объекты перенесены на резервный Сервер
STOP	Объекты возвращены на основной Сервер

Список команд и параметров для объекта **FAILOVER** представлен в таблице:

Команда – описание команды	**Комментарий**
"FORCED_START" – принудительный перенос конфигурации основного Сервера на резервный	Обратный перенос конфигурации выполняется по команде FORCED_STOP или при перезагрузке/ переподключении основного Сервера
"FORCED_STOP" – принудительный перенос конфигурации с резервного Сервера на основной	

10.17 OPERATORPROTOCOL Протокол оператора

Объект **OPERATORPROTOCOL** соответствует системному объекту **Протокол оператора**.

От объекта **OPERATORPROTOCOL** поступают события, представленные в таблице ниже. Запуск процедур происходит при возникновении соответствующего события.

Событие	**Описание события**	**Параметры события**
ACTIVATE_LEFT	Оператор кликнул левой кнопкой мыши по ячейке события в окне **Протокола оператора**	
ACTIVATE_RIGHT	Оператор кликнул правой кнопкой мыши по ячейке события в окне **Протокола оператора**	
POSTPONE_PRESSED	Оператор нажал на кнопку **Отложить**	
CREATE_REPORT	Оператор нажал на кнопку **Сформировать** на вкладке **Создать отчет**	В параметре user_id<> указан идентификатор пользователя. В параметрах initial_date<> и final_date<> указаны выбранные в интерфейсе начальная и конечная даты

Событие	Описание события	Параметры события
RESPONSE_ALARM	Оператор нажал на кнопку **Тревожная ситуация**	objtype<> – Тип объекта objid<> – Идентификатор объекта action<> – Название события в Базе данных alarm_time<> – Время возникновения тревоги
RESPONSE_SUSPECT	Оператор нажал на кнопку **Подозрительная ситуация**	
RESPONSE_FALSE	Оператор нажал на кнопку **Ложное срабатывание**	
ACTIVATE_EVENT	Фокусировка на событии: клик по событию в интерфейсе или переход к нужному событию с помощью клавиатуры	

Список команд и параметров для объекта **OPERATORPROTOCOL** представлен в таблице:

Команда – описание команды	Параметры	Описание параметров
"DEL_ALARM" – удалить тревогу	objtype<>	Тип объекта (например, CAM, GRELE и т.д.)
	objid<>	Идентификатор объекта
	options<>	Возможные значения: • first – удалить первую тревогу • last – удалить последнюю тревогу • all либо пусто – удалить все тревоги
"HIDE_BUTTON" – скрыть кнопки присвоения статуса событию	button<>	Названия кнопок через запятую: • alarm – Тревожная ситуация • suspicious – Подозрительная ситуация • false – Ложное срабатывание Пример задания параметра: `button<alarm,suspicious,false>`

	hide<>	1 – скрыть кнопки, перечисленные в параметра button 0 – отобразить кнопки, перечисленные в параметра button
	objtype<>	Тип объекта
	objaction<>	Тип события
	objid<>	Идентификатор объекта

10.18 EVENT_VIEWER Протокол событий

Объект **EVENT_VIEWER** соответствует системному объекту **Протокол событий**.

От объекта **EVENT_VIEWER** поступают события, представленные в таблице ниже. Запуск процедур происходит при возникновении соответствующего события.

Событие	**Описание события**
SHOW_ON_MAP	Оператор выбрал команду «Показать на карте»
SHOW_VIDEO	Оператор выбрал команду «Показать видео»
SHOW_REPORT	Оператор выбрал команду «Вывести отчет»
CREATE_REPORT	Генерируется, если ключ реестра GenerateEventInsteadOfReport установлен равным 1 и оператор выбрал команду «Вывести отчет». При этом сам отчет не открывается. См. также Справочник ключей реестра

Список команд и параметров для объекта **EVENT_VIEWER** представлен в таблице:

Команда – описание команды	**Параметры**	**Описание параметров**
"UPDATE_VIEW" – установить общий цвет фона и/или текста в окне Протокола событий	`bk_color<>`	Общий цвет фона в 16-ричном виде

	defclr<>	Общий цвет текста 16-ричном виде
"SET_FILTERS" – активировать фильтр Протокола событий	filter0<>,filter1<>,filter2<> и т.д.	Название фильтра, созданного в Протоколе событий. Если нужно указать несколько фильтров, то номера самого параметра (0, 1, 2 и т.д.) должны идти по порядку начиная с filter0<>, пропускать цифры нельзя. См. Примеры скриптов с Протоколом событий

10.19 GATE Видеошлюз

Объект GATE соответствует системному объекту **Видеошлюз.**

От объекта **GATE** поступают события, представленные в таблице ниже. Запуск процедур происходит при возникновении соответствующего события.

События	**Описание событий**	**Комментарий**
GATE_LOW_FPS	Упал темп ввода на шлюзе	
ACTIVE	Шлюз активен	Событие генерируется, когда список работающих камер соответствует списку конфигурации Видеошлюза
INACTIVE	Шлюз неактивен	Событие генерируется, когда нет запроса потоков видео через Видеошлюз
ACTIVE_PART	Частичная работа шлюза	Событие генерируется, когда количество реально работающих камер меньше, чем в списке Видеошлюза

Список команд и параметров для объекта **GATE** представлен в таблице:

Команда – описание команды	Параметры	Описание параметров	Особенности
"START_VIDEO" – включение видеопотока камеры и запуск записи в архив	cam<>	Идентификатор камеры, по которой нужно запустить или остановить запись	Команды работают, даже если в Мониторе не отображается выбранная камера. Команды работают, если в Видеошлюзе включена постоянная запись и запись по активной камере – см. Настройка записи в архив Видеошлюза
"STOP_VIDEO" – остановка видеопотока камеры и записи в архив			

10.20 CAM_VMDA_DETECTOR Детектор VMDA

Объект **CAM_VMDA_DETECTOR** соответствует системному объекту **Детектор VMDA**.

От объекта **CAM_VMDA_DETECTOR** поступают события, представленные в таблице ниже. Запуск процедуры происходит при возникновении соответствующего события.

Событие	Описание события	Параметр	Описание параметра
ALARM	Тревога	native_type<>	Для включения данного параметра необходимо задать следующие ключи реестра: VMDA.determineNoise, VMDA.determineGivenTaken, VMDA.determineHumanCar (см. Справочник ключей реестра). Параметр принимает значения: -1 – неизвестный тип объекта (начальное состояние) 0 – другое 1 – человек или группа людей (в зависимости от значения параметра native_value<>: если 1, то человек, если >1, то группа людей) 2 – машина 3 – шум 4 – принесенный предмет 5 – унесенный предмет

Событие	Описание события	Параметр	Описание параметра
		native_value<>	Для включения данного параметра необходимо задать следующие ключи реестра: VMDA.determineNoise, VMDA.determineGivenTaken, VMDA.determineHumanCar (см. Справочник ключей реестра). Счетчик людей для объекта типа «человек». Позволяет определить число людей в группе. Для остальных типов объектов равен -1
		param0<>	Строковое значение, содержащее параметры события от детектора VMDA
ALARM_END	Конец тревоги		
ARMED	Детектор VMDA поставлен на охрану		
DISARMED	Детектор VMDA снят с охраны		

Список команд и параметров для объекта **CAM_VMDA_DETECTOR** представлен в таблице:

Команда – описание команды	Параметры	Описание параметров
"ARM" – поставить на охрану детектор	-	-
"DISARM" – снять с охраны детектор	-	-

Примечание

Если видеокамеры подключены с помощью ONVIF-Сервера, то события от **Детектора VMDA** и прочих интеллектуальных детекторов будут передаваться как события от встроенных детекторов – см. CAM_IP_DETECTOR Детектор встроенный.

10.21 TITLEVIEWER Поиск по титрам

Объект **TITLEVIEWER** соответствует системному объекту **Поиск по титрам**.

От объекта **TITLEVIEWER** поступают события, представленные в таблице. Запуск процедур происходит при возникновении соответствующего события.

Событие	**Описание события**	**Параметры**	**Описание параметров**	**Комментарий**
GO_VIDEO	Запрос видео	<cam>	Идентификатор камеры, по которой найдены титры	Событие генерируется при двойном клике левой кнопкой мыши на строке, содержащей результат поиска
		<date>	Дата	
		<time>	Время	

10.22 PERSON Пользователь

Объект **PERSON** соответствует системному объекту **Пользователь**.

От объекта **PERSON** поступают события, представленные в таблице. Запуск процедур происходит при возникновении соответствующего события.

Событие	**Описание события**
REGISTERED	Вход пользователя в систему
UNREGISTERED	Выход пользователя из системы

10.23 CAM_FACECAPTURE Детектор лиц

Объект **CAM_FACECAPTURE** соответствует системному объекту **Детектор лиц**.

От объекта **CAM_FACECAPTURE** поступают события, представленные в таблице ниже. Запуск процедур происходит при возникновении соответствующего события.

События	**Описание событий**
FACE_DETECTED	Лицо захвачено
FACE_LEAVE	Лицо потеряно

Список параметров для объекта **CAM_FACECAPTURE** представлен в таблице:

Параметры	**Описание параметров**
owner	Имя сервера, на котором произошел захват\потеря лица
fraction	Миллисекунда захвата/потери лица
module	Модуль, на котором происходит захват
date	Дата захвата/потери лица
guid_pk	ID события (генерируется случайным для каждого события)
core_global	Раздать всем ядрам (оповестить всех)
guid	ID захваченного\потерянного лица (генерируется случайным для каждого события)
time	Время захвата/потери
param0	Аналогично параметру **guid**. Используется для вывода информации в столбце «Доп. Инфо» протокола событий

10.24 IPSTORAGE Внешнее хранилище

Объект **IPSTORAGE** соответствует системному объекту **Внешнее хранилище**.

Список команд и параметров для объекта **IPSTORAGE** представлен в таблице:

Команда – описание команды	**Параметр**	**Описание параметра**	**Комментарий**
"IMPORT" – импорт недостающей части архива за заданный период	cam<>	Идентификатор камеры	Команда применяется в случае, если по каким-то причинам не был выполнен автоматический импорт из внешнего хранилища. *Примечание. Если в момент отправки реакции выполняется импорт, то команда не сработает. Чтобы прервать текущую задачу импорта, необходимо сначала отправить команду UPDATE_TIME*
	datetime_from<>	Дата и время, начиная с которых следует выполнять импорт, в формате <ДД-ММ-ГГ ЧЧ:ММ:СС>	
	datetime_to<>	Дата и время, до которых следует выполнять импорт, в формате <ДД-ММ-ГГ ЧЧ:ММ:СС>	
"UPDATE_TIME" – остановить синхронизацию и задать время последнего импорта из внешнего хранилища в файле Settings.xml	cam<>	Идентификатор камеры	Команда применяется, когда необходимо остановить текущую задачу импорта и выполнить команду IMPORT для синхронизации заданного периода архива
	datetime<>	Дата и время последней синхронизации, которые необходимо установить в файле Settings.xml	

10.25 CAM_TITLE Титрователь

Объект **CAM_TITLE** соответствует системному объекту **Титрователь**.

Список команд и параметров для объекта **CAM_TITLE** представлен в таблице:

Команда – описание команды	**Комментарий**
"REINDEX" – запустить обновление базы банных титров	При любом значении параметра _id_ в команде DoReact() запускается переиндексация базы данных титров. См. также Утилита конвертации базы данных титров Cam_title_updater.exe

10.26 TELEGRAM Telegram бот

Объект **TELEGRAM** соответствует системному объекту **Telegram бот**.

От объекта **TELEGRAM** поступают события, представленные в таблице ниже. Запуск процедуры происходит при возникновении соответствующего события.

Событие	**Описание события**	**Комментарий**
ERROR	Ошибка отправки сообщения	В параметре error<> содержится текстовое описание ошибки

Список команд и параметров для объекта **TELEGRAM** представлен в таблице:

Команда – описание команды	**Параметры**	**Описание параметров**
"SEND" – отослать	text<>	Текст сообщения
	chat_id<>	Идентификатор чата
	bot_id<>	Идентификатор бота
	longtitude<>	Долгота геолокации
	latitude<>	Широта геолокации
	address<>	Текстовый адрес геолокации
"SENDPHOTO" – отослать фото	photo<>	Полный путь к файлу изображения
	caption<>	Подпись к файлу
	chat_id<>	Идентификатор чата
	bot_id<>	Идентификатор бота
	longtitude<>	Долгота геолокации

Команда – описание команды	Параметры	Описание параметров
	latitude<>	Широта геолокации
	address<>	Текстовый адрес геолокации

10.27 CAM_IP_DETECTOR Детектор встроенный

Объект **CAM_IP_DETECTOR** соответствует системному объекту **Детектор встроенный**.

От объекта **CAM_IP_DETECTOR** поступают события, представленные в таблице ниже. Запуск процедур происходит при возникновении соответствующего события.

События	Описание событий	Комментарий
Detected	Событие	Событие поступает при получении метаданных от встроенных детекторов. Например, данное событие отображается при получении данных о температуре тела от тепловизора, и т.п. Также событие поступает, если данные от детекторов передаются с использованием Onvif. В параметре param0<> содержится строковое значение, содержащее параметры события. Передаваемые параметры зависят от встроенного детектора; если используется Onvif, то содержимое параметра можно настроить на вкладке **Настройки событий ONVIF-Сервера** (см. Фильтрация событий ONVIF-Сервера).

Примеры событий от встроенных детекторов:

Пример 1

```
// Событие от тепловизора
Event : CAM_IP_DETECTOR|1|DETECTED|slave_id<QA-T51>,
fraction<16>,owner<QA-T51>,module<video.run>,date<23-04-20>,
guid_pk<{1345DC60-3485-EA11-8A95-B06EBF8119EF}>,core_global<1>,time<10:31:06>,
param0<TargetList:name=TargetList;type=6;TemperatureValue0:37.4;json0:{
   "BeginTime" : "20200423T073058.000000",
   "EndTime" : "20200423T073100.000000",
   "EventClass" : "FaceEvent",
   "Hypotheses" : [
      {
```

```
            "Age" : 0,
            "BestTime" : "20200423T073059.000000",
            "Gender" : "unknown",
            "Rectangle" : [ 0.6380, 0.550, 0.0680, 0.1560 ],
            "TemperatureValue" : 37.40
         }
      ],
      "Id" : 1
}
;>
```

Пример 2

```
// Событие от детектора VMDA
Event:CAM_IP_DETECTOR|1|DETECTED|param0<Comment:ver_type<0>,objtype<SLAVE>,int_obj_id
<1>,module<video.run>,
core_global<1>,_TRANSPORT_ID<>,time<12:22:30>,objaction<PING>,onvif_event<>,
date<30-03-21>,slave_id<DESKTOP-JHRURJJ>,
objid<DESKTOP-JHRURJJ;>,int_obj_id<1>,core_global<1>,
guid_pk<{9A989C70-3991-EB11-BDFF-00155DF96D00}>,slave_id<DESKTOP-339SH3U>,time<12:22:3
0>,_timestamp<7520749>,
fraction<465>,date<30-03-21>,owner<DESKTOP-339SH3U>,module<video.run>
```

10.28 SIP_TERMINAL SIP-терминал

Объект **SIP_TERMINAL** соответствует системному объекту **SIP-терминал**.

От объекта **SIP_TERMINAL** поступают события, представленные в таблице ниже. Запуск процедур происходит при возникновении соответствующего события.

Событие	**Описание**	**Содержание параметра param0<>, отображаемого в поле Доп. инфо в Протоколе событий**
CALL_END	Конец вызова	Номер абонента, который звонил
CALL_END_OPERATOR	Конец связи с оператором	Номера абонентов и продолжительность вызова. Например, если параметр принимает значение "903 to 906 (01:04)", это означает, что абонент 903 звонил абоненту 906, и звонок длился 1 минуту и 4 секунды
CALL_END_DEVICE	Конец связи с устройством	
CALL_END_VIRTUAL	Конец связи с особым номером	

Событие	Описание	Содержание параметра param0<>, отображаемого в поле Доп. инфо в Протоколе событий
CALL_BEGIN	Начало вызова	Номера абонентов: совершающий вызов и тот, кому звонят
CALL_TRYING	Попытка вызова	
CALL_BEGIN_VIRTUAL	Начало вызова особого номера	
CALL_TRYING_VIRTUAL	Попытка вызова особого номера	

Список команд и параметров для объекта SIP_TERMINAL представлен в таблице.

Команда – описание команды	Параметры	Описание параметров
"END_ALL_CALLS" – завершить все звонки на указанном терминале (независимо от того, установлено ли соединение)	-	-

10.29 INC_MANAGER Менеджер инцидентов

Объект INC_MANAGER соответствует системному объекту **Менеджер инцидентов**.

От объекта INC_MANAGER поступают события, представленные в таблице. Запуск процедур происходит при возникновении соответствующего события.

События	Описание событий	Параметры	Описание параметров	Комментарий
CLOSE_CLICK	Нажатие кнопки **Закрыть** в интерфейсе			Событие генерируется, когда в интерфейсе **Менеджера инцидентов** инцидент закрывается без обработки оператором

События	**Описание событий**	**Параметры**	**Описание параметров**	**Комментарий**
CLOSE_ALL_CLICK	Нажатие кнопки **Закрыть все** в интерфейсе			Событие генерируется, когда в интерфейсе **Менеджера инцидентов** все инциденты закрываются без обработки оператором
SELECT	Клик по инциденту в интерфейсе			Событие генерируется, если в интерфейсе **Менеджера инцидентов** оператор кликнул по событию правой или левой кнопкой мыши
ACTIVATE_EVENT	Событие выделено Оператором (клик по событию мышью)	alarm_time<>	Время возникновения события	
		event_guid<>	Идентификатор события (генерируется случайным для каждого события)	
		objtype<>	Тип объекта (например, CAM, GRELE и т.д.)	
		action<>	Тип действия (например, MD_START, DISARM и т.д.)	

10.30 INC_SERVER Сервер инцидентов

Объект INC_SERVER соответствует системному объекту **Сервер инцидентов**.

От объекта INC_SERVER поступают события, представленные в таблице ниже. Запуск процедур происходит при возникновении соответствующего события.

События	**Описание событий**	**Комментарий**
EVENT	Событие (инцидент) взято в обработку в **Менеджере инцидентов** или идёт обработка оператором	Событие генерируется: 1) когда оператор берёт событие в обработку; 2) на каждом шаге обработки события. Параметр **serializeBase64** события содержит JSON с подробной информацией об обрабатываемом событии, в том числе сделанные шаги оператора. Полное логирование всех параметров события включается ключом реестра inc_server_send_full_event (см. Интеллект базовый). При inc_server_send_full_event=1: ▪ в параметре **serializeBase64** будет закодированная строка с полем **SourceMsgBase64**, которое содержит закодированную строку события. Это событие содержит параметр **full_event_base64** с полными параметрами исходного события; ▪ если событие эскалируется, то также будет добавляться дополнительный параметр **inc_server_action<escalated>**. Следует иметь в виду, что включение полного логирования параметров приводит к увеличению размера базы данных событий (по умолчанию события **Сервера инцидентов** хранятся в базе данных Intellect в таблице PROTOCOL_INC_SERVER)

Список команд и параметров для объекта INC_SERVER представлен в таблице:

Команда – описание команды	**Параметры**	**Описание параметров**	**Комментарий**
"UPDATE_STATUS" – изменить статус события (инцидента) в **Менеджере инцидентов**	pks<>	Массив идентификаторов событий	Параметры являются фильтрами и наличие хотя бы одного параметра обязательно. Значения параметра можно указать через разделитель. Это означает, что будет выбрано одно ИЛИ другое значение. Пример: objids<1|2>
	objtypes<>	Типы объектов	
	objids<>	Идентификаторы объектов	
	actions<>	Действия	
	status<>	Статус события: 0 – Ожидает обработки 1 – В работе 2 – Приостановлен 3 – Завершено	
"UPDATE_ESCALATE_STATUS" – изменить статус эскалации события (инцидента) в **Менеджере инцидентов**	escalated<>	Статус эскалации события: 0 - Ожидает обработки (не эскалировано) 1 - Эскалировано	
	pks<>, objtypes<>, objids<>, actions<> те же, что и для UPDATE_STATUS		

10.31 DIALOG Окно запроса оператора

Объект **DIALOG** соответствует системному объекту **Окно запроса оператора**.

Список команд и параметров для объекта **DIALOG** представлен в таблице:

Команда – описание команды	**Параметры**	**Описание параметров**
"SETUP" – настройка окна запроса оператора	x<>	Координата левого верхнего угла (0 - 100)
	y<>	Координата левого верхнего угла (0 - 100)
	allow_move<>	0 – запретить перемещение, 1 – разрешить перемещение
"RUN" – показать окно запроса оператора	-	-
"RUN_MODAL" – запуск окна запроса оператора в модальном режиме	-	-
"CLOSE" – закрывает последнее открытое окно запроса оператора	-	-
"CLOSE_ALL" – закрывает все открытые окна запроса оператора	-	-

10.32 MMS Сервис почтовых сообщений

Объект **MMS** соответствует системному объекту **Сервис почтовых сообщений**.

От объекта **MMS** поступают события, представленные в таблице ниже. Запуск процедуры происходит при возникновении соответствующего события.

Событие	**Описание события**
SET_CONNECTIONS	Список доступных подключений

Список команд и параметров для объекта **MMS** представлен в таблице:

Команда – описание команды	**Параметры**	**Описание параметров**
"SETUP" – настройки для сервиса почтовых сообщений	smtp<>	Адрес SMTP сервера
	connection<>	Тип подключения
	smtp_username<>	Имя пользователя
	smtp_password<>	Пароль
	port<>	Номер порта
	flags<>	Флаги
	name <>	Название объекта
"GET_CONNECTIONS" – получить список доступных подключений	-	-

Свойства объекта **MMS** показаны в таблице:

Свойства объекта MMS	**Описание свойств объекта**
ID<>	Идентификатор объекта
PARENT_ID<>	Идентификатор родительского объекта

10.33 MAIL_MESSAGE Почтовое сообщение

Объект **MAIL_MESSAGE** соответствует системному объекту **Почтовое сообщение**.

От объекта **MAIL_MESSAGE** поступают события, представленные в таблице ниже. Запуск процедуры происходит при возникновении соответствующего события.

Событие	**Описание события**
SEND_ERROR	Ошибка отправки сообщения

Событие	**Описание события**
SENT	Сообщение отправлено

Список команд и параметров для объекта **MAIL_MESSAGE** представлен в таблице:

Команда – описание команды	**Параметры**	**Описание параметров**
"SETUP" – настройки для почтового сообщения	from<>	Адрес отправителя
	to<>	Адрес получателя
	cc<>	Копии
	subject<>	Тема сообщения
	body<>	Тело сообщения
	attachments<>	Приложения. Если требуется приложить к письму несколько файлов, пути к файлам вложений разделяются точкой с запятой
	flags<>	Флаги
	name<>	Имя объекта
	pack<>	Способ упаковки приложений
	is_body_html<>	Указывает, применять ли HTML-разметку при отправке. Возможные значения: 1 и 0
	inline<>	Указывает, отображаются ли вложения только в тексте сообщения (значение 1) или и в тексте, и в разделе «Вложения» (значение 0)
"SEND" – отправка почтового сообщения	-	-

Команда – описание команды	**Параметры**	**Описание параметров**
"SEND_RAW" – отправка почтового сообщения с заданием параметров	аналогично команде SETUP	См. пример в разделе Примеры скриптов на языке JScript

Свойства объекта **MAIL_MESSAGE** показаны в таблице:

Свойства объекта MAIL_MESSAGE	**Описание свойств объекта**
ID<>	Идентификатор объекта
PARENT_ID<>	Идентификатор родительского объекта

10.34 VMS Сервис голосовых сообщений

Объект **VMS** соответствует системному объекту **Сервис голосовых сообщений**.

Список команд и параметров для объекта **VMS** представлен в таблице:

Команда – описание команды	**Параметры**	**Описание параметров**
"SEND" – послать сообщение	modem<>	Название устройства
	pulse<>	Тип набора (0 – тоновый, 1 – импульсный)
	name<>	Имя объекта
	redial_attempts<>	Количество попыток дозвона
	redial_delay<>	Пауза между попытками дозвона
	waitfordialtone<>	Ожидание сигнала линии (0 – нет, 1 – да)
	flags<>	Флаги

Свойства объекта **VMS** показаны в таблице:

Свойства объекта VMS	Описание свойств объекта
ID<>	Идентификатор объекта
PARENT_ID<>	Идентификатор родительского объекта

10.35 GRELE Реле

Объект **GRELE** соответствует системному объекту **Реле**.

От объекта **GRELE** поступают события, представленные в таблице ниже. Запуск процедуры происходит при возникновении соответствующего события.

Событие	Описания события
OFF	Реле выключено
ON	Реле включено
SIGNAL_LOST	Потеря связи

Список команд и параметров для объекта **GRELE** представлен в таблице:

Команда – описание команды	Параметры	Описание параметров
"ON" – включить реле	-	-
"OFF" – выключить реле	-	-
"SETUP" – настройки для реле	chan<>	Номер выхода (0 – 15)
	flags<>	Флаги
	name<>	Имя объекта

Свойства объекта **GRELE** показаны в таблице:

Свойства объекта GRELE	**Описание свойств объекта**
ID<>	Идентификатор объекта
PARENT_ID<>	Идентификатор родительского объекта
REGION_ID<>	Идентификатор региона

Объект **GRELE** может находиться в состояниях, описанных в таблице:

Состояние объекта GRELE	**Описание состояния объекта**
"ON"	Реле включено
"OFF"	Реле выключено
"DETACHED_ON"	Потеря связи, при этом реле было включено
"DETACHED_OFF"	Потеря связи, при этом реле было выключено

10.36 GRAY Луч

Объект **GRAY** соответствует системному объекту **Луч**.

От объекта **GRAY** поступают события, представленные в таблице ниже. Запуск процедуры происходит при возникновении соответствующего события.

Событие	**Описание события**
ALARM	Тревога. Данное событие поступает при размыкании или замыкании луча (в зависимости от настройки объекта), если луч поставлен на охрану. Если луч снят с охраны, поступают события Луч разомкнут и Луч замкнут соответственно
ARM	Луч поставлен на охрану
CONFIRM	Тревога принята

Событие	**Описание события**
DISARM	Луч снят с охраны
NOT_VALID_STATE	Зона не готова
OFF	Луч разомкнут. Данное событие поступает при размыкании луча, если луч снят с охраны
ON	Луч замкнут. Данное событие поступает при замыкании луча, если снят с охраны
SIGNAL_LOST	Потеря связи с лучом

Список команд и параметров для объекта **GRAY** представлен в таблице:

Команда – описание команды	**Параметры**	**Описание параметров**
"ARM" – поставить на охрану луч	-	-
"DISARM" – снять с охраны луч	-	-
“CONFIRM” – принять тревогу	-	-
“SETUP” – настройки для луча	chan<>	Номер входа (0 – 15)
	flags<>	Флаги
	name<>	Имя объекта
	type<>	Тип объекта луч (0 – на замыкание, 1 – на размыкание)

Свойства объекта **GRAY** показаны в таблице:

Свойства объекта GRAY	**Описание свойств объекта**
ID<>	Идентификатор объекта

PARENT_ID<>	Идентификатор родительского объекта
REGION_ID<>	Идентификатор региона

Объект **GRAY** может находиться в состояниях, описанных в таблице:

Состояние объекта GRAY	**Описание состояния объекта**
"ARMED"	Луч поставлен на охрану
"DISARMED"	Луч снят с охраны
"ALARMED"	Тревога
"CONFIRMED"	Тревога принята
"DISARMED_ALARM"	Неготовность
"DETACHED_ARMED"	Произошла потеря связи, когда луч был поставлен на охрану
"DETACHED_DISARM"	Произошла потеря связи, когда луч был снят с охраны
“OFF”	Норма

10.37 VNS Сервис голосового оповещения

Объект **VNS** соответствует системному объекту **Сервис голосового оповещения**.

Список команд и параметров для объекта **VNS** представлен в таблице:

Команда – описание команды	**Параметры**	**Описание параметров**
"SETUP" – настройка сервиса голосового оповещения	card<>	Имя звукового устройства. **Примечание**. Имя карты должно строго соответствовать тому названию, что указано в настройках звуковой карты **Сервиса голосового оповещения** системы *Интеллект*
	level<>	Уровень сигнала. Значение параметра варьируется от 0 до 15. По умолчанию оно равно 8, то есть среднему
	channel<>	Набор звуковых каналов. Возможные значения параметра: 0 – нет звукового канала; 1 – левый канал воспроизведения; 2 – правый канал воспроизведения; 3 – левый и правый канал воспроизведения (оба канала)
	flags<>	Флаги
	ip<>	IP-адрес сетевого устройства
	name<>	Имя объекта
	pass<>	Пароль
	user<>	Имя пользователя
"PLAY" – проигрывание звукового файла	file<>	Полный путь к звуковому файлу в формате .wav (с указанием имени проигрываемого файла, например: C:\Program Files (x86)\Intellect\Wav\cam_alarm_1.wav). **Примечание**. Если указано только имя файла, то путь к нему по умолчанию будет взят с ключа реестра InstallPath в разделе «HKEY_LOCAL_MACHINE\SOFTWARE\ITV\Intellect» (HKEY_LOCAL_MACHINE\Software\Wow6432Node\ITV\Intellect для 64-битной системы), в значении параметра «InstallPath». Также в данном параметре есть возможность указать проигрывание нескольких музыкальных файлов с помощью операции «+»

Команда – описание команды	Параметры	Описание параметров
"STOP" – завершение проигрывания файла	-	-

Свойства объекта **VNS** показаны в таблице:

Свойства объекта VNS	Описание свойства объекта
ID<>	Идентификатор объекта
PARENT_ID<>	Идентификатор родительского объекта

10.38 SMS Сервис коротких сообщений

Объект **SMS** соответствует системному объекту **Сервис коротких сообщений**.

От объекта **SMS** поступают события, представленные в таблице ниже. Запуск процедур происходит при возникновении соответствующего события.

Событие	Описание события	Комментарий
RECEIVE	Получено сообщение	Если событие не поступает при получении сообщения на модем, следует использовать ключ реестра ProcessFromSim (см. Справочник ключей реестра). В параметре сообщения message<> содержится текст присланного сообщения. В параметре phone<> содержится телефон, с которого поступило сообщение, в формате +7XXXXXXXXXX

Список команд и параметров для объекта **SMS** представлен в таблице:

Команда – описание команды	**Параметры**	**Описание параметров**
"SETUP" – настройка сервиса коротких сообщений	device<>	SMS устройство
	flags<>	Флаги
	message<>	Текст сообщения
	name<>	Имя объекта
	phone<>	Номер телефона

Свойства объекта **SMS** показаны в таблице:

Свойства объекта SMS	**Описание свойств объекта**
ID<>	Идентификатор объекта
PARENT_ID<>	Идентификатор родительского объекта

10.39 SSS_WATCHDOG Служба перезагрузки системы

Объект **SSS_WATCHDOG** соответствует системному объекту **Служба перезагрузки системы**.

От объекта **SSS_WATCHDOG** поступают события, представленные в таблице ниже. Запуск процедур происходит при возникновении соответствующего события.

События	**Описание событий**
RESTART_EXCEEDED	Превышено количество перезагрузок модуля
RESTART_PROCESS	Перезагрузка модуля

Список команд и параметров для объекта **SSS_WATCHDOG** представлен в таблице:

Свойства объекта **SSS_WATCHDOG** показаны в таблице:

Свойства объекта SSS_WATCHDOG	Описание свойств объекта
ID<>	Идентификатор объекта
PARENT_ID<>	Идентификатор родительского объекта

10.40 BACNET BacNet

Объект **BACNET** соответствует системному объекту **BacNet**.

От объекта **BACNET** поступают события, представленные в таблице ниже. Запуск процедур происходит при возникновении соответствующего события.

Событие	Описание события
ERROR	Получено сообщение об ошибке
EVENT_OCCURES	Подтверждение получения сообщения
WRITE_OCCURES	Подтверждение выполнения записи
WRITE_RESULT	Результат выполнения записи

Список команд и параметров для объекта **BACNET** представлен в таблице:

Команда – описание команды	Параметры	Описание параметров
"WRITE" – отправить значение в устройство BacNet	bacnet_application_tag<>	Тип данных. Возможные значения: NULL = 0 BOOLEAN = 1 UNSIGNED INT = 2 SIGNED INT = 3 REAL = 4 DOUBLE = 5 OCTET STRING = 6 CHARACTER STRING = 7 BIT STRING = 8

	bacnet_value<>	Значение параметра

	bacnet_objtype<>	Тип объекта:
		`ANALOG INPUT = 0`
		`ANALOG OUTPUT = 1`
		`ANALOG VALUE = 2`
		`BINARY INPUT = 3`
		`BINARY OUTPUT = 4`
		`BINARY VALUE = 5`

	bacnet_instance<>	Уникальный глобальный идентификатор устройства BACnet
	bacnet_property_id<>	Идентификатор свойства
	bacnet_device_id<>	Идентификатор устройства BACnet в системе
"EVENT" – отправить сообщение в устройство BACnet	event_type<>	Тип события
	from_state<>	Перевод из состояния
	to_state<>	Перевод в состояние
	message_text<>	Название события

11 Описание объектной модели в программном комплексе Интеллект

11.1 Объект Core и его встроенные методы

Все методы объекта Core можно использовать в **Скриптах**, а также следующие методы можно использовать в **Программах**:

- Sleep
- DoReact
- DoReactGlobal
- NotifyEvent
- NotifyEventGlobal

Остальные методы объекта Core в **Программах** не используются.

11.2 Объекты MsgObject и Event и их встроенные методы и свойства

Все методы объектов MsgObject и Event используются только при написании **Скриптов**.

11.3 Объект Core и его встроенные методы

11.3.1 Объект Core

Объект **Core** – это ядро системы, глобальный статический объект, реализующий методы, используемые для контроля состояния и управления системными объектами программного комплекса *Интеллект*. Методы объекта **Core** позволяют получать сведения о зарегистрированных системных объектах, генерировать для них реакции, изменять состояния. Объект **Core** реализует дополнительные методы для приостановки выполнения скриптов, их отладки, создания глобальных переменных и обращения к ним.

Объект **Core** не является прототипом, и создание других объектов на его основании (т.е. с использованием объекта **Core** как шаблона) не допускается. Все методы объекта **Core** являются статическими. Таким образом, вызов методов объекта **Core** осуществляется непосредственно из скрипта без обращения к самому объекту **Core**.

11.3.2 Методы GetObject

11.3.2.1 Метод GetObjectName

Метод GetObjectName возвращает название объекта, заданное ему при регистрации в программном комплексе.

Синтаксис обращения к методу:

```
function  GetObjectName(objtype : String, id : String) : String
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Задает системный тип объекта, название которого требуется получить. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.

Пример. При тревоге по любому лучу вызывать диалоговое окно с информационным сообщением: «Тревога по Лучу (название луча, по которому зарегистрирована тревога). Луч подключен к Серверу (название раздела, к которому относится тревожный луч)».

> **Примечание.**
>
> Предварительно необходимо с помощью утилиты *Arpedit.exe* создать диалоговое окно и сохранить его в файле test.dlg в папке <Директория установки ПК *Интеллект*>\Program.

```
if (Event.SourceType == "GRAY" && Event.Action == "ALARM")
{
    var grayid = Event.SourceId;
    var grayname = GetObjectName("GRAY", grayid);
    var compname = GetObjectParentId("GRAY", grayid, "COMPUTER");
    DoReactStr("DIALOG", "test", "CLOSE_ALL","");
    DoReactStr("DIALOG", "test", "RUN","Тревога по Лучу '" + grayname + "'. Луч подключен к Серверу '" + compname + "'.");
}
```

11.3.2.2 Метод GetObjectParam

Метод GetObjectParam возвращает значение заданного параметра системного объекта на момент обращения.

Синтаксис обращения к методу:

```
function  GetObjectParam(objtype : String, id : String, param : String) : String
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Задает тип системного объекта, для которого требуется возвратить значение заданного параметра. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
3. **param** – обязательный аргумент. Соответствует названию параметра, значение которого требуется возвратить. Допустимые значения: тип String, диапазон ограничен допустимыми для заданного объекта параметрами.

Пример. См. пример в разделе Метод SetObjectParam.

11.3.2.3 Метод GetObjectParentId

Метод GetObjectParentId возвращает идентификационный (регистрационный) номер родительского объекта для заданного объекта.

Синтаксис обращения к методу:

```
function  GetObjectParentId(objtype : String, id : String, parent : String) : String
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Задает тип системного объекта, для которого требуется возвратить тип родительского объекта. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
3. **parent** – обязательный аргумент. Задает тип системного объекта, родительского (т.е. старшего в соответствии с иерархией системных объектов) по отношению к объекту заданного

аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.

Пример. При отключении любой из камер системы или прекращении поступления видеосигнала с камеры отправлять почтовое сообщение, зарегистрированное в ПК *Интеллект* под номером 1. Сообщение должно включать тему «Внимание! Отключение камеры» и, в теле сообщения, информацию о номере отключенной камеры и номере сервера, на котором она установлена.

> **Примечание.**
>
> Предполагается, что *Сервис почтовых сообщений* настроен и корректно функционирует.

```
if (Event.SourceType == "CAM" && Event.Action == "DETACH")

{
    var cam_id = Event.SourceId;

    var parent_comp_id = GetObjectParentId("CAM", cam_id, "SLAVE");

    DoReactStr("MAIL_MESSAGE","1","SETUP","from<***@mail.ru>,to<***@mail.ru>,body<
Отключение камеры "+cam_id+" на сервере "+parent_comp_id+">,parent_id<1>,subject<
Внимание! Отключение камеры>,name<Почтовое сообщение 1>,objname<Почтовое сообщение 1>");

    DoReactStr("MAIL_MESSAGE", "1", "SEND", "");
}
```

11.3.2.4 Метод GetObjectState

Метод GetObjectState возвращает состояние системного объекта на момент обращения.

Синтаксис обращения к методу:

```
function  GetObjectState(objtype : String, id : String) : String
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Задает тип системного объекта, состояние которого требуется получить. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.

Пример. При включении реле №1 (например, нажатии кнопки, подключенной к реле №1) поставить на охрану луч №1. При повторном включении реле №1 снять с охраны луч №1.

```
if (Event.SourceType == "GRELE" && Event.SourceId == "1" && Event.Action == "ON")
{
    if(GetObjectState("GRAY", "1")=="DISARM")
    {
        SetObjectState("GRAY", "1", "ARM");
    }
    else
    {
        SetObjectState("GRAY", "1", "DISARM");
    }
}
```

Note

Некоторые типы объектов могут иметь несколько состояний одновременно. Например: ATTACHED|DISARMED или ATTACHED|DISARMED|RECORDER_ON|RECORDING.

11.3.2.5 Метод GetObjectParentType

Метод GetObjectParentType возвращает тип родительского объекта для заданного объекта в соответствии предусмотренной в системе иерархией системных объектов.

Синтаксис обращения к методу:

```
function  GetObjectParentType (objtype : String) : String
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Соответствует типу системного объекта, для которого требуется возвратить тип родительского объекта. Допустимые значения: значения типа String, диапазон ограничен допустимыми в системе типами объектов.

Примечание.

В иерархии системных объектов самым старшим является объект Main. Данный объект является родительским для всех объектов типа Computer («Компьютер»), Screen («Экран») и прочих.

Пример. По запуску макрокоманды №1 отобразить в отладочном окне названия четырех объектов, начиная с зоны детектора, в порядке предусмотренной ПК *Интеллект* иерархии.

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
    var objtype = "CAM_ZONE";
    DebugLogString(objtype);
    for(var i = 1; i<=4; i=i+1)
    {
        objtype = GetObjectParentType(objtype);
        DebugLogString(objtype);
    }
}
```

11.3.2.6 Метод GetObjectIdByParam

Метод GetObjectIdByParam позволяет получить идентификатор объекта, у которого некоторый параметр равен заданному значению. В случае, если таких объектов несколько, возвращается идентификатор первого найденного объекта. В случае, если таких объектов не найдено, возвращается 0.

Синтаксис обращения к методу:

```
function GetObjectIdByParam (obj_type : String, obj_param : String, param_value :
String)
```

Аргументы метода:

1. **obj_type** – обязательный аргумент. Задает тип объекта системы, идентификатор которого требуется получить.
2. **obj_param** – обязательный аргумент. Задает название параметра в базе данных, по значению которого требуется искать объект.
3. **param_value** – обязательный аргумент. Задает требуемое значение параметра объекта.

Пример. Найти камеры, с которых поступает черно-белое изображение, и установить для них параметр **Цвет** (color) равным единице.

```
if (Event.SourceType == "MACRO" && Event.SourceId== "1" && Event.Action == "RUN")
{
  var id = GetObjectIdByParam("CAM","color","0"); //получение идентификатора первого
объекта
  while (id) //пока существуют объекты Камера, с которых поступает черно-белое изображение
  {
     SetObjectParam ("CAM", id, "color", "1"); //изменение параметра Цвет для найденного
объекта
     id = GetObjectIdByParam("CAM","color","0"); //получение идентификатора следующего
объекта
  }
}
```

11.3.2.7 Метод GetObjectChildIds

Метод GetObjectChildIds возвращает идентификационные (регистрационные) номера объектов указанного типа, находящихся в иерархии объектов ниже заданного объекта.

Синтаксис обращения к методу:

```
function  GetObjectParentId(parent : String, id : String, objtype : String) :
String[]
```

Аргументы метода:

1. **parent** – обязательный аргумент. Задает тип системного объекта, дочерние объекты которого требуется узнать. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **parent** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
3. **objtype** – обязательный аргумент. Задает тип системного объекта, являющегося дочерним к типу, заданному аргументом **parent**. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.

Пример. По макрокоманде 1 поставить на охрану все камеры на компьютере WS2.

```
if (Event.SourceType == "MACRO" && Event.Action == "RUN" && Event.SourceId == "1")
{
   var children = GetObjectChildIds("SLAVE","DESKTOP-UBOS6BK","CAM");
   ch=children.split(",");
   for (i=0;i<ch.length; i++ )
   {
     DoReactStr("CAM",ch[i],"ARM","");
   }
}
```

11.3.3 Методы DoReact

11.3.3.1 Метод DoReact

Метод DoReact используется для генерации реакций системных объектов. Метод DoReact отправляет реакцию заданному объекту. При этом реакция передается непосредственно тому ядру, на котором зарегистрирован объект, а не всей системе. В методе DoReact реакция задается объектом **MsgObject**.

Синтаксис обращения к методу:

```
function  DoReact(msgevent : MsgObject)
```

Аргументы метода:

1. **msgevent** – обязательный аргумент. Задает реакцию, отправляемую заданному объекту. Допустимые значения: объекты **MsgObject**, ранее созданные в скрипте.

> **Примечание.**
>
> В программном комплексе *Интеллект* выделяют два типа системных сообщений: события и реакции. События, как правило, несут только информационную нагрузку и используются для рассылки оповещений по всем ядрам программного комплекса *Интеллект*, соединенным между собой при конфигурировании архитектуры. В свою очередь, под реакциями понимаются команды, отправляемые конкретным системным объектам. Реакции передаются непосредственно тем ядрам, на которых зарегистрирован требуемый объект, а не всей системе. Для генерации реакций используются методы DoReactStr и DoReact. Для генерации событий – NotifyEventStr и NotifyEvent.

Пример. По замыканию реле №1 замыкать также реле №2 и 3. По размыканию реле №1 размыкать также реле №2.

```
if (Event.SourceType == "GRELE" && Event.SourceId == "1")
{
    var msgevent = Event.Clone();
    if(Event.Action == "ON")
    {
        msgevent.SourceId = "2";
        DoReact(msgevent);
        msgevent.SourceId = "3";
        DoReact(msgevent);
    }
    if(Event.Action == "OFF")
    {
        msgevent.SourceId = "2";
        DoReact(msgevent);
    }
}
```

11.3.3.2 Метод DoReactStr

Метод DoReactStr используется для генерации реакций системных объектов. Метод DoReactStr отправляет реакцию заданному объекту. При этом реакция передается непосредственно тому ядру, на котором зарегистрирован объект, а не всей системе. В методе DoReactStr реакция задается группой аргументов типа String.

Синтаксис обращения к методу:

```
function  DoReactStr(objtype : String,  id : String,  action : String,  param<value>
[, param<value>] : String)
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Соответствует типу системного объекта, для которого требуется генерировать реакцию. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
3. **action** – обязательный аргумент. Задает реакцию, которую требуется генерировать. Допустимые значения: тип String, диапазон ограничен допустимыми для объекта заданного типа реакциями.
4. **param<value>** – обязательный аргумент. Допускается задание нескольких аргументов данного типа. Соответствует параметру (параметрам) реакции системного объекта.

Синтаксис задания значения одному параметру соответствует строке:

"param<value>", где

param – название параметра;

value – значение параметра.

Синтаксис задания значения нескольким параметрам соответствует строке:

"param1<value1>,param2<value2>...".

Список оформляется через запятую без пробелов. В том случае, если ни один параметр задавать не требуется, при обращении необходимо указать пустую строку, например:

```
DoReactStr("CAM","1","REC","");
```

Допустимые значения аргумента **param**: тип String, диапазон ограничен допустимыми для заданной реакции параметрами. Допустимые значения аргумента **value**: тип String, диапазон зависит от устанавливаемого параметра.

Для всех реакций имеется возможность указать задержку выполнения реакции при помощи параметра delay<>. Задержка указывается в секундах.

Примечание

В программном комплексе *Интеллект* выделяют два типа системных сообщений: события и реакции. События, как правило, несут только информационную нагрузку и используются для рассылки оповещений по всем ядрам программного комплекса *Интеллект*,

соединенным между собой при конфигурировании архитектуры. В свою очередь, под реакциями понимаются команды, отправляемые конкретным системным объектам. Реакции передаются непосредственно тем ядрам, на которых зарегистрирован требуемый объект, а не всей системе. Для генерации реакций используются методы DoReactStr и DoReact. Для генерации событий – NotifyEventStr и NotifyEvent.

Пример 1. При регистрации тревоги по любой из видеокамер системы переводить Монитор №1 в режим отображения одного окна видеонаблюдения (однократер) и выводить в данном окне видеосигнал с камеры, по которой зарегистрирована тревога.

```
if (Event.SourceType == "CAM" &&  Event.Action == "MD_START")
{
    var camid = Event.SourceId;
    DoReactStr("MONITOR","1","ACTIVATE_CAM","cam<"+ camid +">");
    DoReactStr("MONITOR","1","KEY_PRESSED","key<SCREEN.1>");
}
```

Пример 2. При окончании тревоги по любой из видеокамер продолжать запись по ней в течении еще 5 секунд, после чего прекращать запись (аналог режима Дозапись).

```
 if (Event.SourceType == "CAM" &&  Event.Action == "MD_STOP")
{
    var camid = Event.SourceId;
    DoReactStr("CAM",camid,"REC_STOP","delay<5>");
}
```

Пример 3. По макрокоманде 1 включать управление телеметрией при помощи мыши на камере 4, выведенной на монитор 10, по макрокоманде 2 отключать.

```
if (Event.SourceType == "MACRO" &&  Event.Action == "RUN" &&  Event.SourceId == "1")
{
    DoReactStr("MONITOR","10","CONTROL_TELEMETRY","cam<4>,on<1>");
}
if (Event.SourceType == "MACRO" &&  Event.Action == "RUN" &&  Event.SourceId == "2")
{
    DoReactStr("MONITOR","10","CONTROL_TELEMETRY","cam<4>,on<0>");
}
```

11.3.3.3 Метод DoReactSetup

Метод DoReactSetup предназначен для временного изменения параметров системного объекта. Данный метод изменяет только заданные параметры объекта, остальные оставляя без изменения.

Синтаксис обращения к методу:

```
function  DoReactSetup (objtype : String,  id : String, param<value>  [,
param<value>] : String )
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Соответствует типу системного объекта, параметрам которого требуется задать значения. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
3. **param<value>** – обязательный аргумент. Допускается задание нескольких аргументов данного типа. Соответствует параметру (параметрам) реакции системного объекта.

Синтаксис задания значения одному параметру соответствует строке:

“param<value>”, где

param – название параметра;

value – значение параметра.

Синтаксис задания значения нескольким параметрам соответствует строке:

“param1<value1>,param2<value2>…”.

Список оформляется через запятую без пробелов.

Допустимые значения аргумента **param**: значения типа String, диапазон ограничен допустимыми для заданной реакции параметрами. Допустимые значения аргумента **value**: значения типа String, диапазон зависит от устанавливаемого параметра.

Пример. По макрокоманде 1 временно удалить все камеры с первого монитора.

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
    DoReactSetup ("MONITOR","1","REMOVE_ALL","");
}
```

11.3.3.4 Метод DoReactSetupCore

Метод DoReactSetupCore предназначен для изменения параметров системного объекта. Данный метод изменяет только заданные параметры объекта, остальные оставляя без изменения.

Синтаксис обращения к методу:

```
function  DoReactSetupCore(objtype : String,  id : String, param<value>  [,
param<value>] : String )
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Соответствует типу системного объекта, параметрам которого требуется задать значения. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
3. **param<value>** – обязательный аргумент. Допускается задание нескольких аргументов данного типа. Соответствует параметру (параметрам) реакции системного объекта.

Синтаксис задания значения одному параметру соответствует строке:

“param<value>”, где

param – название параметра;

value – значение параметра.

Синтаксис задания значения нескольким параметрам соответствует строке:

“param1<value1>,param2<value2>…”.

Список оформляется через запятую без пробелов.

Допустимые значения аргумента **param**: значения типа String, диапазон ограничен допустимыми для заданной реакции параметрами. Допустимые значения аргумента **value**: значения типа String, диапазон зависит от устанавливаемого параметра.

Пример. По макрокоманде 1 установить камерам №1–4 новые значения параметров номер поворотного устройства (telemetry_id), номер микрофона для синхронной записи (audio_id<>). Значения должны быть на единицу больше, чем номера соответствующих камер.

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
    var i;
    for(i=1; i<=4; i=i+1)
    {
        DoReactSetupCore("CAM", i, "telemetry_id<" + (i+1) +">,audio_id<" + (i+1) +
">");
    }
}
```

11.3.3.5 Метод DoReactGlobal

Метод DoReactGlobal используется для генерации реакций системных объектов. Метод DoReactGlobal отправляет реакцию заданному объекту. При этом реакция передается не только тому ядру, на котором зарегистрирован объект, но и всей системе. В методе DoReactGlobal реакция задается объектом **MsgObject**.

Синтаксис обращения к методу:

```
function  DoReactGlobal(msgevent : MsgObject)
```

Аргументы метода:

1. **msgevent** – обязательный аргумент. Задает реакцию, отправляемую заданному объекту. Допустимые значения: объекты **MsgObject**, ранее созданные в скрипте.

Пример. При выполнении макрокоманды №2 ставить луч №2 на охрану. Команду отправлять по всем ядрам системы в виде реакции для регистрации в Протоколе событий.

```
if (Event.SourceType == "MACRO"&& Event.SourceId == "2" && Event.Action == "RUN")
{
    var msgevent = CreateMsg();
    msgevent.SourceType = "GRAY";
    msgevent.SourceId = "2";
    msgevent.Action = "ARM";
    DoReactGlobal(msgevent);
}
```

11.3.4 Методы NotifyEvent

Методы NotifyEvent используются для генерации системных событий. Они различаются по уровню рассылки событий и сигнатуре метода:

Метод	Рассылка	Сигнатура
NotifyEvent	Непосредственно тому ядру, на котором зарегистрирован объект	Событие задается объектом **MsgObject**
NotifyEventStr	Непосредственно тому ядру, на котором зарегистрирован объект	Событие задается группой аргументов типа String
NotifyEventGlobal	По всем ядрам системы	Событие задается объектом **MsgObject**
NotifyEventGlobalStr	По всем ядрам системы	Событие задается группой аргументов типа String

11.3.4.1 Метод NotifyEvent

Метод NotifyEvent используется для генерации системных событий. При этом генерируемое событие отправляется непосредственно тому ядру, на котором зарегистрирован объект, а не всей системе. В

методе NotifyEvent событие задается объектом **MsgObject** (см. Объекты MsgObject и Event и их встроенные методы и свойства).

Синтаксис обращения к методу:

```
function  NotifyEvent(msgevent : MsgObject)
```

Аргументы метода:

1. **msgevent** – обязательный аргумент. Задает событие, отправляемое в систему. Допустимые значения: объекты **MsgObject**, ранее созданные в скрипте.

> **Примечание.**
>
> В программном комплексе *Интеллект* выделяют два типа системных сообщений: события и реакции. События, как правило, несут только информационную нагрузку и используются для рассылки оповещений по всем ядрам программного комплекса *Интеллект*, соединенным между собой при конфигурировании архитектуры. В свою очередь, под реакциями понимаются команды, отправляемые конкретным системным объектам. Реакции передаются непосредственно тем ядрам, на которых зарегистрирован требуемый объект, а не всей системе. Для генерации реакций используются методы DoReactStr и DoReact. Для генерации событий – NotifyEventStr и NotifyEvent.

Пример. По началу архивации видеозаписей модулем **Долговременный архив** №1 отключается аналоговый выход №1 платы видеоввода №2. Необходимо отправлять в систему команду в виде события для регистрации в **Протоколе событий**.

> **Примечание.**
>
> При выполнении данного скрипта отключение аналогового выхода №1 платы видеоввода №2 не произойдет.

```
if (Event.SourceType == "ARCH" && Event.SourceId == "1" && Event.Action == "ACTIVE")
{
    var msgevent = CreateMsg();
    msgevent.SourceType = "GRABBER";
    msgevent.SourceId = "2";
    msgevent.Action = "MUX1_OFF";
    NotifyEvent(msgevent);
}
```

11.3.4.2 Метод NotifyEventGlobal

Метод NotifyEventGlobal используется для генерации системных событий. При этом генерируемое событие рассылается по всем ядрам системы, соединенным по сети. В методе NotifyEventGlobal событие задается объектом **MsgObject** (см. Объекты MsgObject и Event и их встроенные методы и свойства).

Синтаксис обращения к методу:

```
function  NotifyEventGlobal (msgevent : MsgObject)
```

Аргументы метода:

1. **msgevent** – обязательный аргумент. Задает событие, отправляемое в систему. Допустимые значения: объекты **MsgObject**, ранее созданные в скрипте.

Пример. При выполнении макрокоманды №1 первая камера ставится на запись. Необходимо отправлять команду по всем ядрам системы в виде события для регистрации в Протоколе событий.

> **Примечание.**
>
> При выполнении данного скрипта не произойдет постановки камеры №1 на запись.

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
    var msgevent = CreateMsg();
    msgevent.SourceType = "CAM";
    msgevent.SourceId = "1";
    msgevent.Action = "REC";
    NotifyEventGlobal(msgevent);
}
```

11.3.4.3 Метод NotifyEventStr

Метод NotifyEventStr используется для генерации системных событий. При этом генерируемое событие отправляется непосредственно тому ядру, на котором зарегистрирован объект, а не всей системе. В методе NotifyEventStr событие задается группой аргументов типа String.

Синтаксис обращения к методу:

```
function  NotifyEventStr(objtype : String, id  : String, event : String, param<value>
[, param<value>] : String )
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Соответствует типу системного объекта, для которого требуется генерировать событие. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
3. **event** – обязательный аргумент. Задает событие, которое требуется генерировать. Допустимые значения: тип String, диапазон ограничен допустимыми для объекта заданного типа событиями.
4. **param<value>** – обязательный аргумент. Допускается задание нескольких аргументов данного типа. Соответствует параметру (параметрам) системного события.

Синтаксис задания значения одному параметру соответствует строке:

"param<value>", где

param – название параметра;

value – значение параметра.

Синтаксис задания значения нескольким параметрам соответствует строке:

"param1<value1>,param2<value2>...".

Список оформляется через запятую без пробелов. В том случае, если ни один параметр задавать не требуется, при обращении необходимо указать пустую строку, например:

```
NotifyEventStr("CAM","1","MD_START","");
```

Допустимые значения аргумента **param**: тип String, диапазон ограничен допустимыми для заданного события параметрами. Допустимые значения аргумента **value**: тип String, диапазон зависит от устанавливаемого параметра.

> **Примечание**
>
> В программном комплексе *Интеллект* выделяют два типа системных сообщений: события и реакции. События, как правило, несут только информационную нагрузку и используются для рассылки оповещений по всем ядрам программного комплекса Интеллект, соединенным между собой при конфигурировании архитектуры. В свою очередь, под реакциями понимаются команды, отправляемые конкретным системным объектам. Реакции передаются непосредственно тем ядрам, на которых зарегистрирован требуемый объект, а не всей системе. Для генерации реакций используются методы DoReactStr и DoReact. Для генерации событий – NotifyEventStr и NotifyEvent.

Пример. При регистрации тревоги по камере отправлять в систему событие «тревожное блокирование» для соответствующего камере раздела. Если идентификационный номер тревожной камеры лежит в диапазоне от 1 до 4 – для раздела №1, если от 5 до 10 – для раздела №2.

```
if (Event.SourceType == "CAM" && Event.Action == "MD_START")
{
    var regionid;
    if (Event.SourceId <=4)
    {
        regionid = "1";
    }
    if ((Event.SourceId > 4) &&  (Event.SourceId <= 10))
    {
        regionid = "2";
    }
    NotifyEventStr("REGION", regionid, "PANIC_LOCK", "");
}
```

11.3.4.4 Метод NotifyEventGlobalStr

Метод NotifyEventGlobalStr используется для генерации системных событий. При этом генерируемое событие рассылается по всем ядрам системы, соединенным по сети. В методе NotifyEventGlobalStr событие задается группой аргументов типа String.

Синтаксис обращения к методу:

```
function  NotifyEventGlobalStr(objtype : String, id  : String, event : String,
param<value>  [, param<value>] : String )
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Соответствует типу системного объекта, для которого требуется генерировать событие. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
3. **event** – обязательный аргумент. Задает событие, которое требуется генерировать. Допустимые значения: тип String, диапазон ограничен допустимыми для объекта заданного типа событиями.
4. **param<value>** – обязательный аргумент. Допускается задание нескольких аргументов данного типа. Соответствует параметру (параметрам) системного события.

Синтаксис задания значения одному параметру соответствует строке:

"param<value>", где

param – название параметра;

value – значение параметра.

Синтаксис задания значения нескольким параметрам соответствует строке:

"param1<value1>,param2<value2>...".

Список оформляется через запятую без пробелов. В том случае, если ни один параметр задавать не требуется, при обращении необходимо указать пустую строку, например:

```
NotifyEventGlobalStr("CAM","1","MD_START","");
```

Допустимые значения аргумента **param**: тип String, диапазон ограничен допустимыми для заданного события параметрами. Допустимые значения аргумента **value**: тип String, диапазон зависит от устанавливаемого параметра.

> **Примечание**
>
> В программном комплексе *Интеллект* выделяют два типа системных сообщений: события и реакции. События, как правило, несут только информационную нагрузку и используются для рассылки оповещений по всем ядрам программного комплекса Интеллект, соединенным между собой при конфигурировании архитектуры. В свою очередь, под реакциями понимаются команды, отправляемые конкретным системным объектам. Реакции передаются непосредственно тем ядрам, на которых зарегистрирован требуемый объект, а не всей системе. Для генерации реакций используются методы DoReactStr и DoReact. Для генерации событий – NotifyEventStr и NotifyEvent.

Пример. При регистрации тревоги по камере отправлять всем ядрам системы событие «тревожное блокирование» для соответствующего камере раздела. Если идентификационный номер тревожной камеры лежит в диапазоне от 1 до 4 – для раздела №1, если от 5 до 10 – для раздела №2.

```
if (Event.SourceType == "CAM" && Event.Action == "MD_START")
{
    var regionid;
    if (Event.SourceId <=4)
    {
        regionid = "1";
    }
    if ((Event.SourceId > 4) &&  (Event.SourceId <= 10))
    {
        regionid = "2";
    }
    NotifyEventGlobalStr("REGION", regionid, "PANIC_LOCK", "");
}
```

11.3.5 Метод SetObjectParam

Метод SetObjectParam используется для задания значений параметрам системных объектов.

Синтаксис обращения к методу:

```
function  SetObjectParam(objtype: String,  id: String,  param : String, value :
String)
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Соответствует типу системного объекта, параметрам которого требуется задать значения. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
3. **param** – обязательный аргумент. Соответствует параметру системного объекта. Допустимые значения: тип String, диапазон ограничен допустимыми для заданного объекта параметрами.
4. **value** – обязательный аргумент. Соответствует значению, задаваемому параметру param системного объекта. Допустимые значения: тип String, диапазон зависит от устанавливаемого параметра.

Пример. По запуску Макрокоманды 1 проверять, настроены ли камеры 1–4 на передачу цветного видеосигнала. При обнаружении камеры, настроенной на передачу черно-белого видеосигнала переводить ее в режим работы в цвете (устанавливая ее параметру **Цветность** (“color”) значение **true** (“1”)).

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")

{

    var i;
    for(i=1;i<=4;i=i+1)
    {
        if (GetObjectParam("CAM",i,"color") == "0")
        {
            SetObjectParam("CAM",i,"color","1");
        }
    }
}
```

Примечание.

Если на момент запуска скрипта активен изменяемый в нем объект (т.е. открыта панель его настроек), то изменение параметров объекта методом SetObjectParam не будет произведено. Например, если открыта панель настроек объекта **Камера 1** и запущен вышеприведенный скрипт, режим работы камеры 1 не будет изменен на цветной.

11.3.6 Метод SetObjectState

Метод SetObjectState используется для изменения состояний системных объектов.

Синтаксис обращения к методу:

```
function  SetObjectState(objtype : String, id : String, state : String)
```

Аргументы метода:

1. **objtype** – обязательный аргумент. Соответствует типу системного объекта, состояние которого требуется изменить. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом objtype типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
3. **state** – обязательный аргумент. Соответствует состоянию, в которое требуется перевести объект. Допустимые значения: тип String, диапазон ограничен допустимыми для заданного объекта состояниями.

Пример. Каждый час проверять поставлена ли камера 1 на охрану. В том случае, если камера 1 снята с охраны, поставить ее на охрану.

> **Примечание.**
>
> Предварительно необходимо создать объект **Таймер** с идентификационным номером 1. Установить параметру **Минуты** объекта **Таймер** значение 30. В данном случае таймер будет срабатывать каждый час, например, следующим образом: в 09:30, 10:30, 11:30 и т.д.

```
if (Event.SourceType == "TIMER" && Event.SourceId == "1" && Event.Action == "TRIGGER")
{
    if (GetObjectState("CAM", "1") == "DISARMED")
    {
        SetObjectState("CAM", "1", "ARMED");
    }
}
```

11.3.7 Метод DebugLogString

Метод DebugLogString используется для вывода пользовательских сообщений в отладочные окна утилиты *Редактор-Отладчик*.

Синтаксис обращения к методу:

```
function  DebugLogString(output : String)
```

Аргументы метода:

1. **output** – обязательный аргумент. Задает строку сообщения, которую требуется вывести в отладочное окно утилиты *Редактор-Отладчик*. Допустимые значения: тип String.

Пример. При регистрации в системе какого-либо события от любого из микрофонов выводить его в отладочное окно.

```
if (Event.SourceType == "OLXA_LINE")
{
    var msgstr = Event.MsgToString();
    DebugLogString("Событие от микрофона" + msgstr);
}
```

11.3.8 Метод Base64Decode

Метод Base64Decode используется для декодирования строк, закодированных по схеме Base64.

Синтаксис обращения к методу:

```
function Base64Decode(data_in: String, WideChar: Boolean)
```

Аргументы метода:

1. **data_in** – обязательный аргумент. Задает строку в Base64, которую необходимо декодировать;
2. **WideChar** – обязательный аргумент. Определяет тип кодировки. Возможные значения: 0 или 1. Если тип кодировки Unicode, то значение аргумента 1, иначе 0.

Пример. По запуску макрокоманды №1 декодировать строку, заданную в Base64. Вывести результат декодирования в отладочное окно утилиты *Редактор-Отладчик* (результатом является строка «Intellect JScript»).

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
    var str = Base64Decode("SW50ZWxsZWN0IEpTY3JpcHQ= ", 0);
    DebugLogString(str);
}
```

11.3.9 Метод Sleep

Метод Sleep используется для приостановки выполнения скрипта на заданное время.

Синтаксис обращения к методу:

```
function  Sleep(milliseconds : int)
```

Аргументы метода:

1. **milliseconds** – обязательный аргумент. Задает время, на которое требуется приостановить выполнение скрипта. Указывается в миллисекундах. Допустимые значения: тип int.

Пример 1. По запуску макрокоманды №1 последовательно воспроизводить с помощью аудиопроигрывателя № 1 звуковые файлы cam_alarm_1.wav, cam_alarm_2.wav, cam_alarm_3.wav из папки …\Intellect\Wav\. Задержка между началом воспроизведения каждого последующего звукового файла должна составлять 5 секунд (5000 миллисекунд).

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
    var i;
    for(i=1; i<=3; i=i+1)
    {
        DoReactStr("PLAYER", "1", "PLAY_WAV", "file<\cam_alarm_" + i + ".wav>");
        Sleep(5000);
    }
}
```

Пример 2. По запуску макрокоманды №2 запускать таймер №1, срабатывающий через каждые 10 секунд в течение 1 минуты с момента запуска макрокоманды №2.

Примечание.

Для запуска данного скрипта необходимо предварительно создать объект **Таймер** с идентификационным номером 1. Параметры объекта следует оставить установленными по умолчанию (**Любой(-ая)**). Объект **Таймер 1** может быть отключен.

```
if (Event.SourceType == "MACRO" && Event.SourceId == "2" && Event.Action == "RUN")

{
    for(i=0; i<=5; i=i+1)

    {
        DoReactStr("TIMER","1", "DISABLE", "");
        Sleep(10000);
        DoReactStr("TIMER","1", "ENABLE", "");

        NotifyEventStr("TIMER","1", "TRIGGER", "");
    }
    DoReactStr("TIMER","1", "DISABLE", "");
}
```

11.3.10 Метод Itv_var

Метод **Itv_var** используется для задания и возвращения значений глобальных переменных.

Синтаксис обращения к методу:

```
function  Itv_var (globalvar : String) : String
```

Globalvar – обязательный аргумент. Задает название глобальной переменной. Допустимые значения: тип String, удовлетворяющие требованиям к допустимым названиям строковых (String) параметров системного реестра ОС Windows.

> **Примечание.**
>
> Глобальные переменные хранятся в системном реестре, что обеспечивает сохранность их значений после перезапуска ОС Windows. Все глобальные переменные хранятся в ветвях реестра HKEY_USERS\S-1-5-21-...\Software\VMSScript\VMSSCRIPT и HKEY_CURRENT_USER\Software\VMSScript\VMSSCRIPT. Для доступа к глобальной переменной непосредственно из реестра нужно поискать по ее названию.

Пример. По запуску макрокоманды №1 сохранять значение параметра **Яркость** ("bright") для камеры №10 в глобальную переменную cam10bright. По запуску макрокоманды №2 устанавливать камерам 1-4 значение параметра **Яркость** равным значению глобальной переменной cam10bright.

```
if (Event.SourceType == "MACRO" && Event.Action == "RUN")

{
    if(Event.SourceId == "1")
    {
        Itv_var("cam10bright") = GetObjectParam("CAM", "10", "bright");
```

```
    }
    if (Event.SourceId == "2")
    {
        var cam10bright = Itv_var("cam10bright");
        for(i=1; i<=4; i=i+1)
        {
            SetObjectParam("CAM", i, "bright", cam10bright);
        }
    }
}
```

11.3.11 Метод Int_var

Метод **Int_var** используется для задания и возвращения значений глобальных переменных целочисленного типа.

> **Внимание!**
>
> Метод **Int_var** использует то же хранилище, что и Метод Itv_var, но приводит тип переменной к целочисленному.

Синтаксис обращения к методу:

```
function  Int_var (globalvar : String) : int
```

Globalvar – обязательный аргумент. Задает название глобальной переменной. Допустимые значения: тип String, удовлетворяющие требованиям к допустимым названиям строковых (String) параметров системного реестра ОС Windows.

> **Примечание.**
>
> Глобальные переменные хранятся в системном реестре, что обеспечивает сохранность их значений после перезапуска ОС Windows. Все глобальные переменные хранятся в ветвях реестра HKEY_USERS\S-1-5-21-...\Software\VMSScript\VMSSCRIPT и HKEY_CURRENT_USER\Software\VMSScript\VMSSCRIPT. Для доступа к глобальной переменной непосредственно из реестра нужно поискать по ее названию.

Пример. В приведенном ниже тестовом примере для проверки работы метода глобальной переменной с названием "2" присваивается значение 1, которое затем увеличивается на единицу и выводится в отладочное окно скрипта.

```
if(Event.Action =="RUN")
```

```
{
    Int_var(2) = 1;
    Int_var("2")++;
    DebugLogString(Int_var("2").toString());
}
```

11.3.12 Метод GetIPAddress

Метод GetIPAddress возвращает IP-адрес соединения ядер программного комплекса *Интеллект* в соответствии с существующей архитектурой распределенной системы видеонаблюдения.

Синтаксис обращения к методу:

```
function  GetIPAddress (dst : String, src : String) : String
```

Аргументы метода:

1. **dst** – обязательный аргумент. Задает наименование удаленного компьютера, на котором установлено ядро программного комплекса *Интеллект*. Значение аргумента **dst** должно совпадать с одним из наименований компьютеров, зарегистрированных при настройке архитектуры распределенной системы видеонаблюдения. Допустимые значения: значения типа String, удовлетворяющие требованиям к сетевым именам компьютеров; диапазон ограничен зарегистрированными в системе наименованиями компьютеров.
2. **src** – обязательный аргумент. Задает наименование локального компьютера (компьютера, с которого производится запуск скрипта). Значение аргумента **src** должно совпадать наименованием локального компьютера, под которым он зарегистрирован в программном комплексе *Интеллект*. Допустимые значения: значения типа String, удовлетворяющие требованиям к сетевым именам компьютеров.

> **Примечание.**
>
> Данные обо всех зарегистрированных при настройке распределенной архитектуры соединениях локального компьютера (ядра) с другими удаленными компьютерами (ядрами) отображаются во вкладке **Архитектура** диалогового окна **Настройка системы**.

Пример. По тревоге от камеры определить имя компьютера, к которому подключена данная камера, и вывести в отладочное окно IP-адрес соединения данного компьютера с локальным (на котором работает скрипт).

```
if (Event.SourceType == "CAM" && Event.Action == "MD_START")

{
    var camid = Event.SourceId;
```

```
    var compname = GetObjectParentId("CAM", camid, "COMPUTER");\\определение имени компьютера, к которому подключена тревожная камера
    var ip = GetIPAddress(compname,"WS1"); \\ определение IP-адреса соединения с компьютером, на котором установлена тревожная камера
    DebugLogString("IP-адрес соединения с компьютером, на котором установлена тревожная камера
  " + ip);
}
```

Примечание.

Вместо “WS1” в примере необходимо вписать имя компьютера, на котором запускается скрипт и установлено ядро ПК *Интеллект*.

11.3.13 Метод CreateMsg

Метод CreateMsg предназначен для создания объектов на основании прототипа **MsgObject** (см. Объекты MsgObject и Event).

Синтаксис обращения к методу:

```
function CreateMsg() : MsgObject
```

Аргументы метода отсутствуют.

Пример 1. При регистрации тревоги по камере отправлять в систему событие «тревожное блокирование» для соответствующего раздела. Если идентификационный номер тревожной камеры лежит в диапазоне от 1 до 4 – для раздела №1, если от 5 до 10 – для раздела №2.

```
if (Event.SourceType == "CAM" && Event.Action == "MD_START")
{

var msgevent = CreateMsg();
    msgevent.SourceType = "REGION";
    msgevent.Action = "PANIC_LOCK";
    if (Event.SourceId <=4)
    {
        msgevent.SourceId = "1";
    }
    if ((Event.SourceId > 4) &&  (Event.SourceId < 10))
    {
        msgevent.SourceId = "2";
    }
    NotifyEvent(msgevent);
}
```

Пример 2. При запуске таймера №1 через каждые 30 секунд запускать макрокоманду №1.

Примечание.

Для запуска данного скрипта необходимо предварительно создать объект **Таймер** с идентификационным номером 1. Установить параметру **Секунда** объекта **Таймер** значение 1, остальные параметры оставить без изменений (по умолчанию **Любой(ая)**).

```
if (Event.SourceType == "TIMER" && Event.SourceId == "1" && Event.Action == "TRIGGER")
{
    var msg = CreateMsg();
    msg.StringToMsg(GetObjectParams("TIMER", "1"));
    if(msg.GetParam("s") == "1")
    {
        DoReactStr("MACRO", "1", "RUN", "");
        SetObjectParam("TIMER","1","s","30");
        DoReactStr("TIMER","1", "DISABLE", "");
        DoReactStr("TIMER","1", "ENABLE", "");
    }
    if(msg.GetParam("s") == "30")
    {
        DoReactStr("MACRO", "1", "RUN", "");
        SetObjectParam("TIMER","1","s","1");
        DoReactStr("TIMER","1", "DISABLE", "");
        DoReactStr("TIMER","1", "ENABLE", "");
    }
}
```

11.3.14 Методы Lock и Unlock

Методы Lock и Unlock используются для создания глобальной критической секции при необходимости обеспечения синхронизации скриптов, запускаемых в отдельных потоках. Метод Lock открывает критическую секцию, метод Unlock закрывает.

Внимание!

Необходимо обязательно вызывать метод Unlock, если был вызван метод Lock. В противном случае система может зависнуть.

Рекомендуется по возможности избегать использования методов Lock и Unlock.

Синтаксис обращения к методам:

```
function Lock()
```

```
function Unlock()
```

Пример. По макрокоманде 1 посчитать общее количество лучей и реле, находящихся в тревоге. Подсчет объектов каждого типа производить одновременно (в отдельном скрипте). Результат записать в глобальную переменную counter.

Скрипт 1:

```
//Считается количество реле в тревоге
var i = Number(0);
if (Event.SourceType == "MACRO" && Event.SourceId== "1" && Event.Action == "RUN")
{
    var msg = CreateMsg();
    msg.StringToMsg(GetObjectIds("GRELE"));
    var objCount = msg.GetParam("id.count");
    var k;
    for(k= 0; k < objCount; k++)
        if(GetObjectState("GRELE", msg.GetParam("id." + k))== "ALARM"){
            Lock();
            i = Itv_var("counter");
            i++;
            Itv_var("counter")=i;
            Unlock();
        }
}
```

Скрипт 2:

```
//Считается количество лучей в тревоге
var i = Number(0);
if (Event.SourceType == "MACRO" && Event.SourceId== "1" && Event.Action == "RUN")
{
    var msg = CreateMsg();
    msg.StringToMsg(GetObjectIds("GRAY"));
    var objCount = msg.GetParam("id.count");
    var k;
    for(k = 0; k < objCount; k++)
        if(GetObjectState("GRAY", msg.GetParam("id." + k))== "ALARMED"){
            Lock();
            i = Itv_var("counter");
            i++;
            Itv_var("counter")=i;
            Unlock();
    }
}
```

> **Примечание.**
>
> Если в данном примере не использовать методы Lock() и Unlock(), могут возникнуть коллизии и посчитанное значение окажется меньше, чем реальное.

11.3.15 Метод IsAvailableObject

Метод IsAvailableObject используется для определения текущих прав доступа к объекту.

Синтаксис обращения к методу:

```
function  IsAvailableObject(compname: String, objtype: String,  id: String,  param :
String) : String
```

Метод возвращает 0, если текущему пользователю не назначены права типа **param** на доступ к объекту, и 1, если назначены.

Аргументы метода:

1. **compname** – обязательный аргумент. Соответствует имени объекта **Компьютер**, на базе которого создан объект в дереве оборудования.
2. **objtype** – обязательный аргумент. Соответствует типу системного объекта, права доступа к которому требуется выяснить. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
3. **id** – обязательный аргумент. Соответствует идентификационному (регистрационному) номеру объекта заданного аргументом **objtype** типа. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе идентификационными номерами объектов заданного типа.
4. **param** – обязательный аргумент. Соответствует номеру типа прав, наличие которых требуется выяснить. Описание прав приведено в разделе Ограничение прав администрирования, управления и мониторинга документа Руководство Администратора. Допустимые значения:
   a. 0 – права доступа rightsNoView. Метод вернет 1, если нет прав на администрирование, управление и мониторинг объекта (красный крестик).
   b. 1 – права доступа rightsNoControl. Метод вернет 1, если есть права только на мониторинг объекта (буква М).
   c. 2 – права доступа rightsViewAndControl. Метод вернет 1, если есть права на управление и мониторинг объекта (флажок зеленого цвета).
   d. 3 – права доступа rightsViewOrControl. Метод вернет 1, если есть права на мониторинг или управление объектом.
   e. 4 – права доступа rightsNot.
   f. 5 – права доступа rightsConfigure. Метод вернет 1, если есть права на администрирование, управление и мониторинг объекта (флажок серого цвета).

Пример. В дереве оборудования на базе объекта **Компьютер** с именем "Comp" создан объект **Камера** с идентификатором 1. Выяснить текущие права на доступ к объекту.

```
var i = 0;
for(i = 0; i <= 5; i++)
{
     var result =
    IsAvailableObject('Comp','CAM','1', i);
    DebugLogString("right "+i+" = "+result);
}
```

11.3.16 Метод GetUserId

Метод GetUserId возвращает идентификатор текущего пользователя ПК *Интеллект*.

Синтаксис обращения к методу:

```
function  GetUserId (cmp : String) : String
```

Аргументы метода:

1. **cmp** – обязательный аргумент. Задает имя компьютера, на котором установлен программный комплекс *Интеллект*. Допустимые значения: значения типа String, удовлетворяющие требованиям к сетевым именам компьютеров; диапазон ограничен зарегистрированными в системе наименованиями компьютеров.

Пример. Вывести в отладочном окне идентификатор текущего пользователя программного комплекса *Интеллект,* установленного на компьютере 'WS3'.

```
DebugLogString(GetUserId("WS3"));
```

11.3.17 Метод GetEventDescription

Метод GetEventDescription используется для получения описания события на естественном языке.

Синтаксис обращения к методу:

```
function  GetEventDescription (obj_type : String, event : String)
```

Аргументы метода:

1. **obj_type** – обязательный аргумент. Задает тип объекта системы, описание события которого требуется получить.
2. **event** – обязательный аргумент. Задает название события, описание которого требуется получить.

Пример. При получении событий по камере 1 выводить сообщения об этом на естественном языке в отладочное окно.

```
if (Event.SourceType == "CAM"&& Event.SourceId == "1")
{
var str = GetEventDescription("CAM", Event.Action);
DebugLogString(str);
}
```

11.3.18 Метод SaveToFile

Метод SaveToFile используется для сохранения в файл кадра с камеры, который поступает в параметре data события FRAME_SENT.

Синтаксис обращения к методу:

```
function  SaveToFile (path: String,  data: String,  param : Boolean)
```

Сохранение кадра также может осуществляться при помощи реакции GET_FRAME объекта CAM. Для этого необходимо указать в параметре path данной реакции путь для сохранения файла с кадром. Событие FRAME_SENT создается в системе, если у реакции GET_FRAME отсутствует параметр path. При этом в параметре data события FRAME_SENT хранится кадр видеоизображения, который требуется сохранить при помощи метода SaveToFile.

Данная реакция позволяет экспортировать кадр видеоизображения, даже если камера не отображается в окне Монитора видеонаблюдения.

Аргументы метода:

1. **path** – обязательный аргумент. Задает полный путь для сохранения файла с кадром.
2. **data** – обязательный аргумент. Задает данные для сохранения в файл.
3. **param** – обязательный аргумент. Определяет необходимость перекодирования из формата base64 перед сохранением. Возможные значения параметра:
    a. true – перед сохранением декодировать из base64;
    b. false – сохранить строку без перекодировки.

Время выполнения сохранения кадра зависит от частоты опорных кадров. Чем больше частота опорных кадров, тем меньше время.

Пример. В случае поступления кадра с Камеры 1 сохранить его в файл test.jpg на диске D:

```
if (Event.SourceType == "CAM" && Event.SourceId == "1" && Event.Action ==
"FRAME_SENT")
  {
    SaveToFile("D:\\test.jpg",Event.GetParam("data"),true);
  }
```

11.3.19 Метод GetLinkedObjects

Метод GetLinkedObjects используется для получения списка объектов, привязанных к заданной камере при помощи объекта **Связь объектов** (см. Связь объектов с камерами).

Синтаксис обращения к методу:

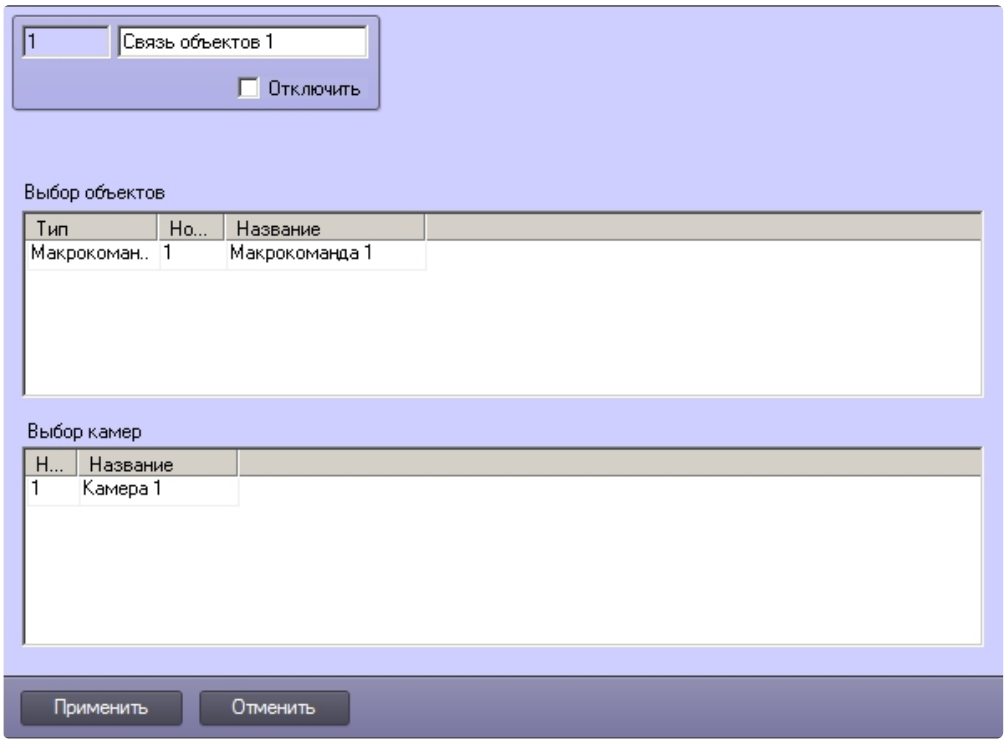
```
function  GetLinkedObjects (type1 : string, id : string, type2 : string)
```

Аргументы метода:

1. **type1** – тип объекта, для которого следует вернуть связанные.
2. **id** – идентификатор объекта, для которого следует вернуть связанные.
3. **type2** – тип связанных объектов, которые следует вернуть. Если передана пустая строка, будут возвращены связанные объекты всех типов.

Пример.

Объект **Связь объектов** настроен следующим образом:



Вывести в отладочное окно скрипта список объектов, связанных с камерой 1.

```
if (Event.SourceType == "MACRO")
{
    var msgstr = GetLinkedObjects("CAM","1","MACRO")
    DebugLogString("Связанные объекты  " + msgstr);
```

```
}
```

В результате в отладочное окно скрипта будет выведена строка «Связанные объекты MACRO:1».

11.3.20 Метод WriteIni

Метод WriteIni используется для записи строковой переменной в ini-файл.

Синтаксис обращения к методу:

```
function WriteIni(varName: String, varValue: String, path: String)
```

Аргументы метода:

1. **varName** – обязательный аргумент. Задает имя переменной для хранения в файле.
2. **varValue** – обязательный аргумент. Задает значение переменной.
3. **path** – обязательный аргумент. Задает полный путь к файлу ini, в котором должна храниться переменная. Хранилище переменных можно поместить на сетевом ресурсе, для этого в данном аргументе следует задать сетевой путь.

Пример. Записать переменную MyVar в файл \\fileserver\temp\test.ini, задав ей значение "Hello world!". Затем считать записанное значение и вывести его в отладочное окно скрипта.

```
WriteIni("MyVar", "Hello world", "\\\\fileserver\\temp\\test.ini");
var result = ReadIni("MyVar","\\\\fileserver\\temp\\test.ini");
DebugLogString(result);
```

11.3.21 Метод ReadIni

Метод ReadIni используется для чтения значения строковой переменной из ini-файла.

Синтаксис обращения к методу:

```
function ReadIni (varName: String, path: String): String
```

Аргументы метода:

1. **varName** – обязательный аргумент. Задает имя переменной, хранящейся в файле.
2. **path** – обязательный аргумент. Задает полный путь к файлу ini, в котором хранится переменная.

Пример см. в разделе Метод WriteIni.

11.3.22 Метод AddIni

Метод AddIni используется для записи, изменения и чтения значения целочисленной переменной из ini-файла. Метод возвращает значение переменной, полученное после изменения.

Синтаксис обращения к методу:

```
function AddIni(varName: String, varValue: int, path: String): int
```

1. **varName** – обязательный аргумент. Задает имя переменной в файле.
2. **varValue** – обязательный аргумент. Задает значение переменной либо значение, которое следует добавить к существующему значению переменной:
    a. Если в файле хранится переменная с именем varName и строковым значением, переменной будет присвоено значение varValue.
    b. Если в файле нет переменной с именем varName, будет создана такая переменная, и ей будет присвоено значение varValue.
    c. Если в файле хранится переменная с именем varName, которая имеет целочисленное значение, или же значение ее приводится к целочисленному типу, то значение будет приведено, и к нему будет прибавлено varValue.
3. **path** – обязательный аргумент. Задает полный путь к файлу ini, в котором должна храниться переменная. Хранилище переменных можно поместить на сетевом ресурсе, для этого следует задать сетевой путь.

Пример. В файле "C:\test.ini" нет переменной "MyVar". Записать в данный файл такую переменную со значением -1, прибавить к ней 1 и вывести полученное значение в отладочное окно скрипта.

```
var result = AddIni("MyVar", -1, "C:\\test.ini");

result = AddIni("MyVar", 1, "C:\\test.ini");

DebugLogString(result);
```

11.3.23 Метод SetTimer

Метод SetTimer используется для запуска таймера.

Синтаксис обращения к методу:

```
function SetTimer (id : int, milliseconds : int)
```

Аргументы метода:

1. **id** – обязательный аргумент. Задает идентификатор таймера. Допустимые значения: тип int, string.

2. **milliseconds** – обязательный аргумент. Задает период, с которым будет срабатывать таймер, если его не остановить методом KillTimer(). Указывается в миллисекундах. Допустимые значения: тип int.

Пример. Через 2 секунды после выполнения макрокоманды 1 начинать запись по камере 1.

```
if(Event.SourceType=="LOCAL_TIMER" && Event.Action=="TRIGGERED" &&
Event.SourceId==333) // можно указать Event.SourceId == "333", т.е. использовать тип
идентификатора string
{
  var actuallyKilled = KillTimer(333);
  if(actuallyKilled == 1)
  {
     DoReactStr("CAM","1","REC","");
  }
}

if(Event.SourceType=="MACRO"&& Event.SourceId == "1" && Event.Action == "RUN")
{
  SetTimer(333,2000); //333 - id, 2000 msec = 2 sec - period
}
```

11.3.24 Метод KillTimer

Метод KillTimer используется для остановки таймера. Возвращает 1, если в результате выполнения функции таймер был остановлен.

Синтаксис обращения к методу:

```
function KillTimer (id : int) : int
```

Аргументы метода:

id – обязательный аргумент. Задает идентификатор таймера. Допустимые значения: тип int, string.

Пример см. в разделе Метод SetTimer.

11.3.25 Метод Base64EncodeFile

Метод Base64EncodeFile используется для кодирования файлов по схеме Base64. Возвращает строку.

См. также Метод Base64Decode и Метод SaveToFile.

Синтаксис обращения к методу:

```
function Base64EncodeFile (data_in: String): String
```

Аргумент метода:

data_in – обязательный аргумент. Задает путь к файлу, который необходимо закодировать.

Пример. По запуску макрокоманды №1 закодировать файл 1.bmp в Base64 и записать в файл 2.bmp.

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
    var s = Base64EncodeFile("d:\\1.bmp");
    SaveToFile("d:\\2.bmp",s, true);
}
```

11.3.26 Метод Base64EncodeW

Метод Base64EncodeW используется для кодирования Unicode-строки по схеме Base64. Возвращает строку.

См. также Метод Base64Decode.

Синтаксис обращения к методам:

```
function Base64EncodeW (data_in: String): String
```

Аргумент метода:

data_in – обязательный аргумент. Задает строку, которую необходимо закодировать.

Пример. Декодировать строку из Unicode в Base64, закодировать обратно и вывести в отладочную строку.

```
var test = Base64Decode("MAAzAC0AMAA3AC0AMgAwADEAOQAgADEANQA6ADMAOQA6ADQAMAA6AA0ACgB0
AGUAcwB0ACAAMQANAAoAMAAzAC0AMAA3AC0AMgAwADEAOQAgADEANQA6ADQAMgA6ADIAMQA6AA0ACgB0AGUAc
wB0ACAAMgA=",true);
DebugLogString("----->"+test);
var res = Base64EncodeW(test);
DebugLogString("----->"+res);
```

Если методу Base64Decode передан параметр true, то вывод скрипта будет следующим:

```
03-07-2019 15:39:40:
test 1
03-07-2019 15:42:21:
test 2
```

11.3.27 Методы run_cmd и run_cmd_timeout

Метод run_cmd используется для выполнения команд в командной строке из скрипта.

Метод run_cmd_timeout используется для выполнения команд в командной строке с заданием таймаута завершения процесса.

При вызове команд окно командной строки не открывается, команды выполняются в скрытом режиме.

Синтаксис обращения к методам:

```
function run_cmd (cmd: String)
function run_cmd_timeout (cmd: String, timeout: int)
```

Аргументы методов:

1. **cmd** – команда для командной строки.
2. **timeout** – только для run_cmd_timeout, таймаут завершения процесса командной строки после выполнения команды.

Пример 1. Запустить утилиту curl и отправить POST-запрос с текстом "Hello" на тестовый URL https://postman-echo.com/post.

```
var s = run_cmd("curl --request POST --url https://postman-echo.com/post --data
\'Hello\'");
DebugLogString(s);
```

Пример 2. Отображать загрузку CPU на График 1, обновляя информацию по Таймеру 3.

```
var id = "1"; // идентификатор объекта Графики
var timer_id = "3"; // идентификатор объекта Таймер
slave_id = "DESKTOP-5397BVV"; // идентификатор объекта Компьютер

if (Event.SourceType == "TIMER" && Event.Action == "TRIGGER" && Event.SourceId ==
timer_id)
{
    var date = Event.GetParam("date");
    var time = Event.GetParam("time");
    var cpu = "for /f \"tokens=2* delims=^,\" %k in ('typeperf \"\\Processor
Information(_Total)\\% Processor Time\" -sc 1 ^| findstr \":\"') do echo %k";
    var cpu_usage = run_cmd(cpu);
    var cpu_usage2 = cpu_usage.replace(/\"/g,"");
    var cpu_usage3 = cpu_usage2.replace(/\s/g,"");
    DebugLogString(cpu_usage3);
    DoReactStr("ANALOGCHART",id,"ANALOG_PARAMS","int_obj_id<"+id+">,parent_id<>,slave
_id<"+slave_id+">,objid<"+id+">,chan<5>,core_global<1>,text<"+cpu_usage3+">,
min_val<0>,max_val<100>,sensor_id<cpu_usage>,time<"+time+">,date<"+date+">");
}
```

11.3.28 Метод WriteIniAny

Метод WriteIniAny используется для записи строковой переменной в ini-файл. В отличие от метода WriteIni, в WriteIniAny можно задать для записи необходимую секцию файла.

Синтаксис обращения к методу:

```
function WriteIniAny(varName: String, varValue: String, path: String, section:
String)
```

Аргументы метода:

1. **varName** – обязательный аргумент. Задает имя переменной для хранения в файле.
2. **varValue** – обязательный аргумент. Задает значение переменной.
3. **path** – обязательный аргумент. Задает полный путь к файлу ini, в котором должна храниться переменная. Хранилище переменных можно поместить на сетевом ресурсе, для этого в данном аргументе следует задать сетевой путь.
4. **section** – обязательный аргумент. Задает название секции ini-файла, в которую необходимо записать переменную.

Пример. Записать переменную MyVar в секцию "config" файла C:\\Backup\\test2.ini, задав ей значение "Hello world!". Затем считать записанное значение и вывести его в отладочное окно скрипта.

```
WriteIniAny("MyVar", "Hello world!", "C:\\Backup\\test2.ini", "config");
var result = ReadIniAny("MyVar","C:\\Backup\\test2.ini", "config");
DebugLogString(result);
```

11.3.29 Метод ReadIniAny

Метод ReadIniAny используется для чтения значения строковой переменной из ini-файла. В отличие от метода ReadIni, в ReadIniAny можно задать секцию файла, из которой необходимо прочитать переменную.

Синтаксис обращения к методу:

```
function ReadIniAny (varName: String, path: String, section: String): String
```

Аргументы метода:

1. **varName** – обязательный аргумент. Задает имя переменной, хранящейся в файле.
2. **path** – обязательный аргумент. Задает полный путь к файлу ini, в котором хранится переменная.
3. **section** – обязательный аргумент. Задает название секции ini-файла, из которой необходимо прочитать переменную.

Пример см. на странице Метод WriteIniAny.

11.3.30 Метод AddIniAny

Метод AddIniAny используется для записи, изменения и чтения значения целочисленной переменной из ini-файла. В отличие от метода AddIni, в AddIniAny можно задать секцию файла, в которой находится целочисленная переменная. Метод возвращает значение переменной, полученное после изменения.

Синтаксис обращения к методу:

```
function AddIniAny(varName: String, varValue: int, path: String, section: String):
int
```

1. **varName** – обязательный аргумент. Задает имя переменной в файле.
2. **varValue** – обязательный аргумент. Задает значение переменной либо значение, которое следует добавить к существующему значению переменной:
    a. Если в файле хранится переменная с именем varName и строковым значением, переменной будет присвоено значение varValue.
    b. Если в файле нет переменной с именем varName, будет создана такая переменная, и ей будет присвоено значение varValue.
    c. Если в файле хранится переменная с именем varName, которая имеет целочисленное значение, или же значение ее приводится к целочисленному типу, то значение будет приведено, и к нему будет прибавлено varValue.
3. **path** – обязательный аргумент. Задает полный путь к файлу ini, в котором должна храниться переменная. Хранилище переменных можно поместить на сетевом ресурсе, для этого следует задать сетевой путь.
4. **section** – обязательный аргумент. Задает название секции ini-файла, в которой находится переменная.

Пример. В секции "config" файла "C:\test.ini" нет переменной "MyVar". Записать в секцию данного файла такую переменную со значением -1, прибавить к ней 1 и вывести полученное значение в отладочное окно скрипта.

```
var result = AddIniAny("MyVar", -1, "C:\\test.ini", "config");

result = AddIniAny("MyVar", 1, "C:\\test.ini", "config");

DebugLogString(result);
```

11.4 Объекты MsgObject и Event и их встроенные методы и свойства

11.4.1 Объекты MsgObject и Event

Объект **MsgObject** – это прототип (шаблон), используемый для создания объектов и реализующий используемые для обработки системных событий программного комплекса *Интеллект* методы и свойства. Методы и свойства объекта **MsgObject** позволяют получать сведения о системных объектах, от которых поступают события (или для которых инициализируются события), генерировать реакции для системных объектов, изменять их состояния и тому подобное.

Обращение к методам и свойствам объекта-прототипа **MsgObject** осуществляется через объявленные и инициализированные на его основе объекты или через статический объект **Event**.

Объект **Event** – это статический объект, реализующий интерфейс обращения к системным событиям программного комплекса *Интеллект*. Объект **Event** обеспечивает доступ к системному событию, по которому был осуществлен запуск скрипта. При работе с объектом **Event** доступны все методы и свойства прототипа **MsgObject**.

Объявление (создание) объектов на основе прототипа **MsgObject** выполняется с использованием метода CreateMsg базового объекта **Core** (см. Метод CreateMsg).

11.4.2 Метод GetSourceType

Метод GetSourceType возвращает системный тип объекта **MsgObject** или **Event**.

Синтаксис обращения к методу:

```
function  GetSourceType() : String
```

Аргументы метода отсутствуют.

Пример. По макрокоманде № 1 ставить на охрану для камер № 1–4 зоны детекторов № *.1, настроенные на работу в режиме **День**. По макрокоманде № 2 ставить на охрану для камер № 1–4 зоны детекторов № *.2, настроенные на работу в режиме **Ночь**. По макрокоманде № 3 ставить на охрану для камер № 1–4 зоны детекторов № *.3, настроенные на работу в режиме **Осадки**.

> **Примечание.**
>
> Значок "*" соответствует идентификационному номеру видеокамеры в системе (от 1 до 4).

```
if(Event.GetSourceType() == "MACRO" && Event.GetAction() == "RUN")
{
    var k;
    //Перевод камер в режим работы  "День"  путем постановки на охрану зон детекторов №  *.1
```

```
    if(Event.GetSourceId() == "1")
    {
        for(k=1; k<=4; k=k+1)
        {
            DoReactStr("CAM_ZONE", k + ".1", "ARM", "");
            DoReactStr("CAM_ZONE", k + ".2", "DISARM", "");
            DoReactStr("CAM_ZONE", k + ".3", "DISARM", "");
        }
    }
    //Перевод камер в режим работы  "Ночь"  путем постановки на охрану зон детекторов №  *.2
    if(Event.GetSourceId() == "2")
    {
        for(k = 1; k <= 4; k = k+1)
        {
            DoReactStr("CAM_ZONE", k + ".1", "DISARM", "");
            DoReactStr("CAM_ZONE", k + ".2", "ARM", "");
            DoReactStr("CAM_ZONE", k + ".3", "DISARM", "");
        }
    }
    //Перевод камер в режим работы  "Осадки"  путем постановки на охрану зон детекторов №  *.3
    if(Event.GetSourceId() == "3")
    {
        for(k = 1; k <= 4; k = k+1)
        {
            DoReactStr("CAM_ZONE", k + ".1", "DISARM", "");
            DoReactStr("CAM_ZONE", k + ".2", "DISARM", "");
            DoReactStr("CAM_ZONE", k + ".3", "ARM", "");
        }
    }
}
```

11.4.3 Метод GetSourceId

Метод GetSourceId возвращает идентификационный (регистрационный) номер объекта **MsgObject** или **Event**.

Синтаксис обращения к методу:

```
function  GetSourceId() : String
```

Аргументы метода отсутствуют.

Пример см. в разделе Метод GetSourceType.

11.4.4 Метод GetAction

Метод GetAction возвращает событие, поступившее в форме объекта **Event**, или заданное для объекта **MsgObject**.

Синтаксис обращения к методу:

```
function  GetAction() : String
```

Аргументы метода отсутствуют.

Пример см. в разделе Метод GetSourceType.

11.4.5 Метод GetParam

Метод GetParam возвращает значение заданного параметра системного объекта для объекта **MsgObject** или **Event**.

Синтаксис обращения к методу:

```
function  GetParam(param: String) : String
```

Аргумент метода:

param – обязательный аргумент. Соответствует названию параметра системного объекта, для которого создан объект **MsgObject** (или статический объект **Event**). Допустимые значения: тип String, диапазон ограничен допустимыми для объекта заданного типа параметрами.

> **Примечание.**
>
> Если у объекта отсутствует параметр с таким названием, метод возвращает пустую строку.

Пример. При регистрации любого события от любой камеры проверять, с какого компьютера пришло данное событие. В том случае, если имя компьютера равно «WS3», создать копию события, в которой установить имя компьютера равным «Computer».

```
if (Event.SourceType == "CAM")
{
    var msg = Event.Clone();
    if (msg.GetParam("slave_id") == "WS3")
    {
        msg.SetParam("slave_id", "Computer");
        NotifyEvent(msg);
    }
}
```

11.4.6 Метод SetParam

Метод SetParam устанавливает значение заданному параметру объекта **MsgObject** или **Event**. Данный метод изменяет только заданные параметры объекта, остальные оставляя без изменения.

Синтаксис обращения к методу:

```
function  SetParam(param : String,  value : String)
```

Аргументы метода:

1. **param** – обязательный аргумент. Соответствует названию параметра системного объекта, для которого создан объект **MsgObject** (или статический объект **Event**). Допустимые значения: тип String, диапазон ограничен допустимыми для объекта заданного типа параметрами.
2. **value** – обязательный аргумент. Устанавливает значение параметру, заданному аргументом **param**. Допустимые значения: тип String, диапазон зависит от устанавливаемого параметра.

Пример см. в разделе Метод GetParam.

11.4.7 Метод MsgToString

Метод MsgToString преобразует объекты **MsgObject** (в том числе статический объект **Event**) в переменную типа String.

Синтаксис обращения к методу:

```
function  MsgToString() : String
```

Аргументы метода отсутствуют.

Пример. Отправлять сообщения обо всех событиях, зарегистрированных для микрофона №1, на заданный электронный почтовый ящик.

> **Примечание.**
>
> Предполагается, что **Сервис почтовых сообщений** настроен и корректно функционирует.

```
if (Event.SourceType == "OLXA_LINE" && Event.SourceId == "1")
{
    var msgstr = Event.MsgToString();
    DoReactStr("MAIL_MESSAGE", "1", "SETUP", "subject<Микрофон 1>,body<" + msgstr +
">");
    DoReactStr("MAIL_MESSAGE", "1", "SEND", "");
}
```

11.4.8 Метод StringToMsg

Метод StringToMsg преобразует переменную типа String в объект **MsgObject**.

Синтаксис обращения к методу:

```
function StringToMsg(msg : String) : MsgObject
```

Аргумент метода:

msg – обязательный аргумент. Задает переменную типа String, которую требуется преобразовать в объект **MsgObject**. Допустимые значения: переменные типа String, удовлетворяющие синтаксису представления объектов **MsgObject**:

"objtype|id|action|param1<value1>,param2<value2>...", где

- **objtype** – тип системного объекта;
- **id** – идентификационный номер системного объекта;
- **action** – событие или реакция для системного объекта;
- **param1<value1>,param2<value2>** – список параметров со значениями. Список оформляется через запятую без пробелов. Если ни один параметр задавать не требуется, необходимо оставить пустой строку после вертикальной черты «|», например: "CAM|1|MD_START|"

Пример. По тревоге от лучей № 1 и 3 начинать запись аудиосигнала с микрофона № 1. По тревоге от лучей № 2 и 4 начинать запись аудиосигнала с микрофона № 2.

```
if (Event.SourceType == "GRAY" && Event.Action == "ALARM")
{
    var audioid;
    if (Event.SourceId == "1" || Event.SourceId == "3")
    {
        audioid = "1";
    }
    if (Event.SourceId == "2" || Event.SourceId == "4")
    {
        audioid = "2";
    }
    var str = "OLXA_LINE|"+audioid+"|ARM|";
    var msg = CreateMsg();
    msg.StringToMsg(str);
    NotifyEvent(msg);
}
```

11.4.9 Метод StringToParams

Метод StringToParams преобразует переменную типа String в список параметров и перезаписывает существующий список параметров объекта **MsgObject**.

Синтаксис обращения к методу:

```
function StringToParams(params: String)
```

Аргумент метода:

params – Обязательный аргумент. Задает переменную типа String, которую требуется преобразовать в список параметров объекта **MsgObject**. Допустимые значения: переменные типа String, удовлетворяющие синтаксису представления списка параметров объектов **MsgObject**:

“param1<value1>,param2<value2>…”, где

param1<value1>,param2<value2> – список параметров со значениями. Список оформляется через запятую без пробелов. Если ни один параметр задавать не требуется, необходимо оставить пустой строку после вертикальной черты «|», например: “CAM|1|MD_START|”

Пример. При регистрации события **Подключение** («attach») для любой из камер, требуется повторно инициировать событие **Подключение** в системе с измененными значениями параметров **Номер поворотного устройства** (telemetry_id) и **Номер микрофона для синхронной записи** (audio_id). Значения должны быть на единицу больше, чем номера соответствующих камер.

```
if (Event.SourceType == "CAM" && Event.Action == "ATTACH")
{
    var i;
    for (i=1;i<=4;i=i+1)
    {
        var msg = Event.Clone();
        var str = "telemetry_id<" + (i+1) +">,audio_id<" + (i+1) + ">";
        msg.StringToParams(str);
        NotifyEvent(msg);
    }
}
```

11.4.10 Метод Clone

Метод Clone создает копию объектов **MsgObject** и **Event**.

Синтаксис обращения к методу:

```
function Clone() : MsgObject
```

Аргументы метода отсутствуют.

Пример. По замыканию реле №1 начинать видеозапись по камере №1 и замыкать реле №2. По размыканию реле №1 начинать видеозапись по камере №2 и размыкать реле №2.

```
if (Event.SourceType == "GRELE" && Event.SourceId == "1")
{
    var msgevent = Event.Clone();
    if(Event.Action == "ON")
    {
        msgevent.SourceId = "2";
        DoReactStr("CAM","1","REC","");
        DoReactStr("GRELE","2","ON","");
    }
    if(Event.Action == "OFF")
```

```
    {
        msgevent.SourceId = "2";
        DoReactStr("CAM","2","REC","");
        DoReactStr("GRELE","2","OFF","");
    }
}
```

11.4.11 Метод GetObjectIds

Метод GetObjectIds отвечает за получение идентификаторов всех объектов определённого типа.

Синтаксис обращения к методу:

```
function GetObjectIds(objectType : String)
```

Аргумент метода:

objectType – обязательный аргумент. Задает тип системного объекта, для которого требуется возвратить значение заданного параметра (**CAM**, **GRAY**, **GRABBER** и т.п.). Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.

В ответ возвращается строка:

CAM||COUNT|id.3<5>,id.count<4>,id.0<2>,id.1<3>,id.2<4>

где

- id.count<> – количество ID объектов,
- id.[число]<> – ID объекта.

Пример. По запуску Макрокоманды №1 необходимо поставить все камеры на охрану.

```
if (Event.SourceType == "MACRO" && Event.SourceId && Event.Action == "RUN")
{
    var msg = CreateMsg();
    msg.StringToMsg(GetObjectIds("CAM"));
    var objCount = msg.GetParam("id.count");
    var i;
    for(i = 0; i < objCount; i++)
    {
        DoReactStr("CAM", msg.GetParam("id." + i), "ARM", "");
    }
}
```

11.4.12 Метод GetObjectParams

Метод GetObjectParams используется для получения параметров объекта.

Синтаксис обращения к методу:

```
function GetObjectParams(objectType : String, objectId : String)
```

Аргументы метода:

1. **objectType** – обязательный аргумент. Задает тип системного объекта (**CAM**, **GRAY**, **GRABBER** и т.п.), для которого требуется возвратить тип родительского объекта. Допустимые значения: тип String, диапазон ограничен зарегистрированными в системе типами объектов.
2. **objectId** – идентификатор объекта. Допустимые значения: тип String.

Пример. По запуску Макрокоманды №1 необходимо проверить цветность камеры №2. В том случае, если камера №2 цветная, поставить ее в режим записи.

```
if (Event.SourceType == "MACRO" && Event.SourceId && Event.Action == "RUN")
{
    var msg = CreateMsg();
    msg.StringToMsg(GetObjectParams("CAM", "2"));
    if(msg.GetParam("color") == "1")
    {
        DoReactStr("CAM", "2", "REC", "");
    }
}
```

11.4.13 Свойство SourceType

Свойство SourceType позволяет возвращать и устанавливать системный тип для объектов **MsgObject** и **Event**.

Синтаксис обращения к свойству:

```
SourceType : String
```

Пример. По замыканию реле № 1 (например, по нажатию подключенной к реле кнопки) печатать кадры с камер №1 и 2.

```
if (Event.SourceType == "GRELE" && Event.SourceId == "1" && Event.Action == "ON")
{
    //активирование окна видеонаблюдения камеры №1
    DoReactStr("MONITOR","1","ACTIVATE_CAM", "cam<1>");
    //печать кадра, экспортированного с камеры №1
    DoReactStr("MONITOR","1","KEY_PRESSED","key<PRINT>");
    //активирование окна видеонаблюдения камеры №2
    DoReactStr("MONITOR","1","ACTIVATE_CAM", "cam<2>");
    //печать кадра, экспортированного с камеры №2
    DoReactStr("MONITOR","1","KEY_PRESSED","key<PRINT>");
}
```

11.4.14 Свойство SourceId

Свойство SourceId позволяет возвращать и устанавливать идентификационный номер для объектов **MsgObject** и **Event**.

Синтаксис обращения к свойству:

```
SourceId : String
```

Пример см. в разделе Свойство SourceType.

11.4.15 Свойство Action

Свойство Action позволяет возвращать и устанавливать реакцию или событие для объектов **MsgObject** и **Event**.

Синтаксис обращения к свойству:

```
SourceId : String
```

Пример см. в разделе Свойство SourceType.

12 Заключение

Более подробная информация о программном комплексе *Интеллект* содержится в следующих документах:

1. Руководство администратора: настройка системных объектов в интерфейсе.
2. Руководство оператора: работа с ПК *Интеллект*.
3. Руководство по установке и настройке компонентов охранной системы: установка и настройка периферии (камеры, сигнализация, системы контроля доступа и т.д.).

Если в процессе работы с данным программным продуктом у вас возникли трудности или проблемы, вы можете связаться с нами. Однако рекомендуем предварительно сформулировать ответы на следующие вопросы:

1. В чем именно заключается проблема?
2. Когда и после чего появилась данная проблема?
3. В каких именно условиях проявляется проблема?

Чем более полную и подробную информацию вы предоставите, тем быстрее наши специалисты смогут устранить проблему.

Мы всегда работаем над улучшением качества своей продукции, поэтому будем рады любым вашим предложениям и замечаниям, касающимся работы нашего программного обеспечения, а также документации к нему.