Guide for creating scripts (programming)

Guide for creating scripts. Introduction

The Program object. Programming using the embedded language of Axxon PSIM

The Script object. Programming using the JScript language

Description of events and reactions of system objects

Description of the object model in Axxon PSIM

Programming guide. Conclusion

Export to PDF

Guide for creating scripts. Introduction

You can use programming with the help of scripts if *Axxon PSIM* interface settings of objects or Macros capabilities aren't enough to implement any operating scenario of *Axxon PSIM*. For more details about interactions between objects, see the table below.

There are two options for writing scripts in *Axxon PSIM*:

1. **In the embedded programming language**. You can use the **Program** system object of the **Programming** tab—see The Program object. Programming using the embedded language of Axxon PSIM.
2. **In the JScript language**. You can use the **Script** system object of the **Programming** tab—see The Script object. Programming using the JScript language.

Both options work, but the **Program** object is deprecated and is no longer developed. We recommend using the **Script** object and the JScript language to write scripts.

This guide had the following information:

- description of the settings for both objects,
- syntax of the scripts and their debugging,
- examples of scripts for each language.

The scripts use:

- events, reactions and commands of *Axxon PSIM* objects. The main ones are described in Description of events and reactions of system objects.
- the Core, MsgObject, Event objects and their embedded methods described in Description of the object model in Axxon PSIM.

Methods of setting logical interactions between objects in Axxon PSIM

Axxon PSIM functionality is based on logical interactions between objects. General information on methods of setting logical interactions:

Method of setting logical interaction	**Description**	**Implementation**	**Example**
Setting panels of system objects	Basic configuration of interaction between system objects	Implemented using functionality of system objects—see Axxon PSIM configuration and setup	Configuring video display from the **Camera** in the **M onitor** interface window
Macro	Configuration of simple interactions between objects if basic object settings are insufficient	Implemented using the **Macro** object—see Creating and using macros	Enabling actuator (relay) when sensor is closed
Program	Configuration of complex interactions between objects if functionality of the **Macro** object is insufficient	Implemented using the **Program** object as the code in the embedded programming language of *Axxon PSIM*—see The Program object. Programming using the embedded language of Axxon PSIM	Return PTZ cameras to their original position and take a photo every 15 minutes
Script		Implemented using the **Script** object as JScript code—see The Script object. Programming using the JScript language	

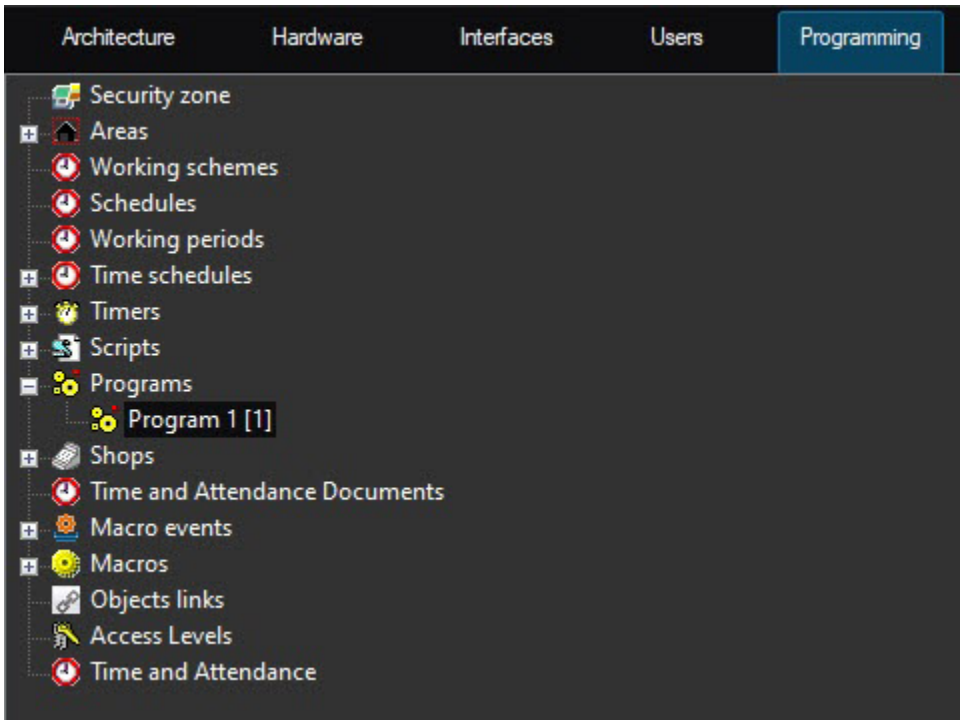
The Program object. Programming using the embedded language of Axxon PSIM

Programming tools in Axxon PSIM

The Program system object

The **Program** system object is used to initialize the program written in the *Axxon PSIM* programming language in *Axxon PSIM* and set its parameters.

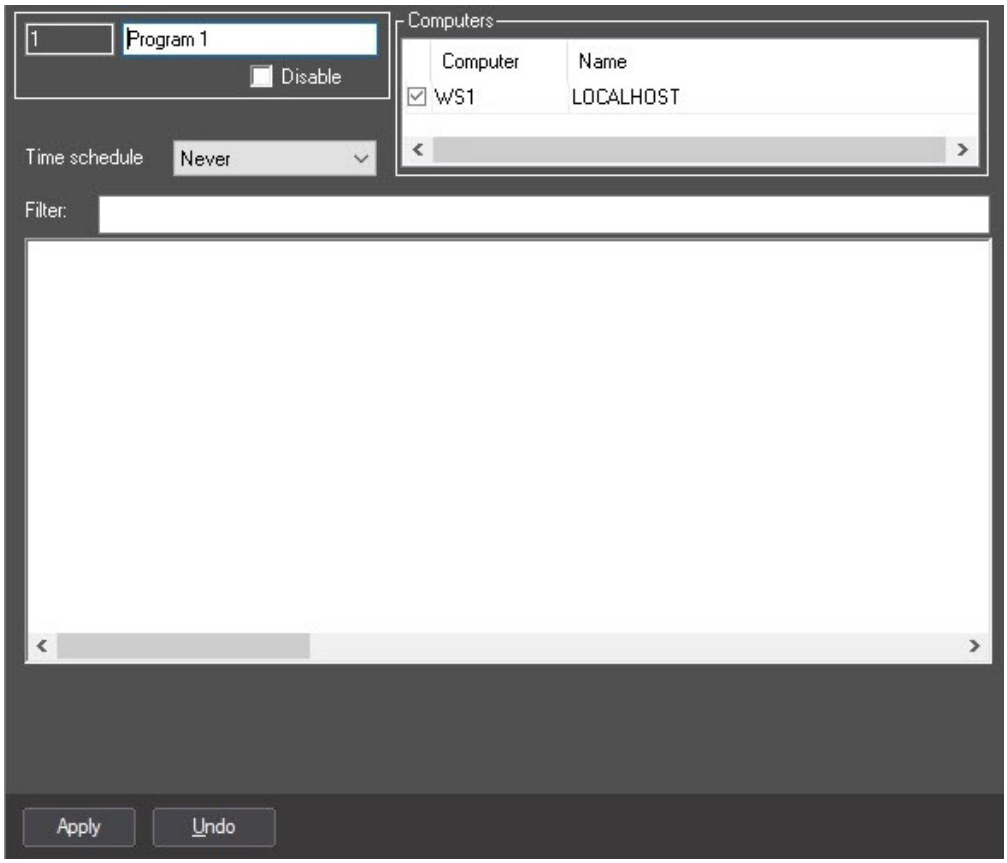
The **Program** system object is created on the basis of the **Programs** object on the **Programming** tab of the **System settings** dialog window.



Attention!

Creation of more than 100 **Program** system objects can cause system instability.

See the settings panel of the **Program** system object in the figure below:



On the settings panel of the **Program** system object, specify the time schedule of program execution and computers (cores) on which the program must be executed.

> **Note**
>
> To set all checkboxes next to all computers, select a cell in the column with checkboxes and press Ctrl+A. To clear all checkboxes, select a cell and press Shift+A.

To pre-filter events processed by the program, set the value in the **Filter** field. The filter format is TYPE|ID|EVENT divided by semicolon. For example, CAM||MD_STOP;CAM||MD_START to filter **Alarm** and **Alarm end** events from all **Camera** objects.

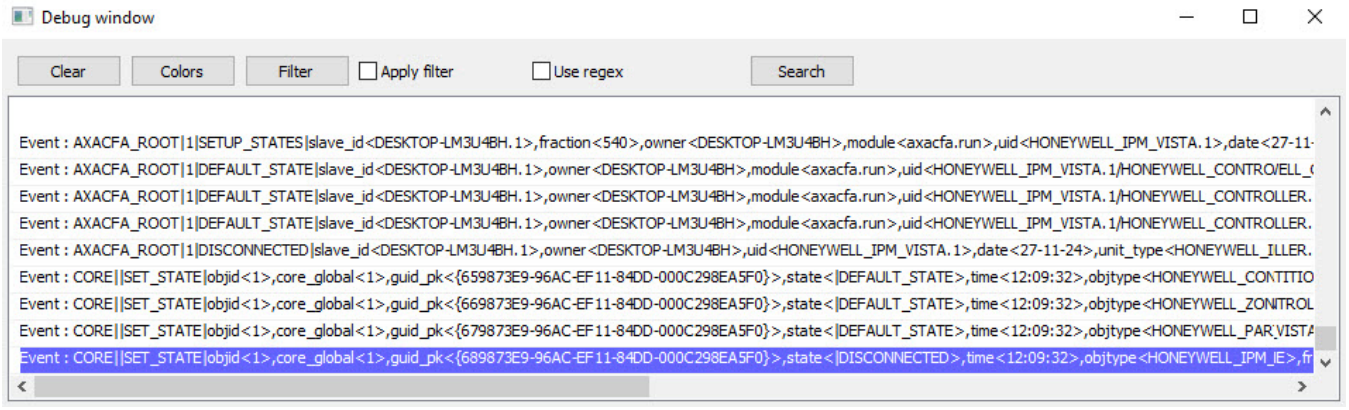
There is the word processor on the settings panel of the **Program** system object. It is used for writing and editing the program code.

You can undo or redo some action using keyboard shortcuts in the word processor on the settings panel of the **Program** system object. To undo some action, press **Alt**+**Backspace**, to redo, press **Ctrl**+**Y**.

Debug window

The Debug window is used to view data about all events registered in the system.

You can call the **Debug window** by using the **Debug** command in the **Run** menu on the Main control panel. The Debug window is displayed at the bottom of the screen.



By default, the **Debug window** opens in the Debug 4 debug mode. You can change the debug mode of the **Debug window** using the *tweaki.exe* utility (see The Debug window) or using the **DebugLevel** registry key (see Registry keys reference guide. For more information on working with the registry, see Working with Windows OS registry).

Syntax analyser

Embedded syntax analyser enables spell check of basic registered words, such as OnEvent, DoReact, OnTime, Wait, Sleep, and so on. These registered words are marked in black in the program text field. Note that the analyser doesn't check if command parameters are written correctly, so you must be very attentive in these cases.

```
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+">");
```

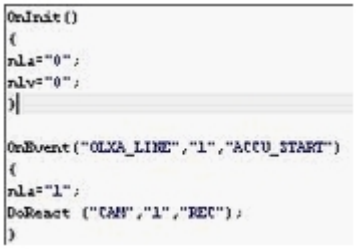
To change the font size, use the key combinations:

- **CTRL** and **+** to make font larger

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

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

- **CTRL** and **–** to make font smaller



Recommended procedure of writing programs

On the page:

1. Set a general task.
2. Outline the task into subtasks.
3. Write subtasks and debug them.
4. Find and fix bugs.

Setting a general task

You must have a clear vision of what must happen in the system when certain event occur. Specify the ID of devices that participate in generating events and actions.

Outlining the task into subtasks

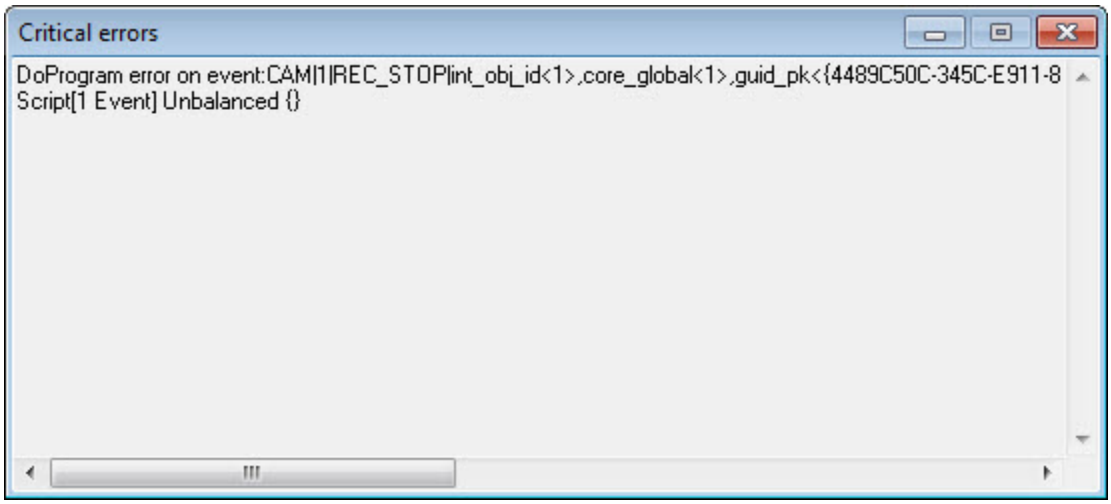
If several events must be processed in one task, then it must be clear what to do with each event. If possible, exclude the possibility of loop script execution, it means, exclude any recursive actions if they are not related to task execution.

Writing subtasks and debugging them

The most difficult part of writing scripts is creating the list of actions with possible use of logic and cycle operations. Debugging of this part of programming takes much time. Events generation that needs processing is not usually easy-to-use, especially on a real object—for example fire sensor triggering or motion by camera that is far programming place (from the server with the system core). In this case, it is recommended to generate an event manually at the stage of debugging, the best way is to run an empty macro. After the body of the script is debugged, there is a real event instead of running the empty macro. Moreover, you can make sure whether the event is written correctly events without starting the action list by running an empty macro and watching its performance in the Debug window.

Finding and fixing bugs

At program startup, embedded syntax analyzer checks if names of functions are spelled correctly, but does not check the program syntax (position of key characters: commas, semicolons and nested parentheses). In order to track the bugs in the program, if any, you must activate the **Debug 4** debug mode (see Enabling and configuring the debug mode of Axxon PSIM). In there are syntax errors, the **Critical errors** window will be displayed at the stage of the program body execution. This window lists the names of functions with incorrect syntax and other debugging information.



Note

If the syntax is correct, but the program still doesn't work or works with errors, we recommend rewriting the program as a script in JScript (see The Script object. Programming using the JScript language).

Description of syntax

Script consists of the set of procedures.

All operators executed inside procedures are enclosed in {..} blocks.

If you want to write a comment, you need to put reserved characters // before the comment.

Description of variables

All variables in the system are string variables.

To compare string variables and values, use the bool strequal (string1,string2) function. The strequal function returns the nonzero value if strings are equal (see Description of functions).

To do integer operations, use the str(string1) function (see Description of functions).

Description of procedures

Standard procedures

There are three standard procedures that can be performed when the corresponding event occurs:

1. **OnInit()**—used for initialization of variables (setting initial values) that will be used in scripts. It is executed before starting all modules of the system. We recommend calling the procedure once for all scripts.
   Example of use:

```
OnInit(){
  flag=1;
  num=8; //variables will be initialized at startup
}
```

2. **OnTime**(DOW (1-7), day-month-year, hours, minutes, seconds)—running at specific time.

```
OnTime(W,D,X,Y,H,C,S)
{
//W - DOW (0 - Monday, 6 - Sunday);
//D - date in the day-month-year format, 16 August 2001 is "16-08-01"
//X,Y - reserved
  //H - hour
  //C - minutes
  //S - seconds
  // COMPARING WITH PARAMETERS, THE ACTION IS SPECIFIED FURTHER
}
```

   Examples of use:

```
OnTime(W,"16-08-01",X,Y,"11","11","30")
{
 //the code will be executed on 16 August, 2001 at 11:11:30
}
```

```
OnTime(W,D,X,Y,"11","11","30")
{
 //the code will be executed every day at 11:11:30
}
```

```
OnTime(W,"16-08-01",X,Y,H,C,S)
{
 //the code will be executed on 16 August, 2001
 //every second
}
```

```
OnTime(W,"16-08-01",X,Y,"11","11",S)
{
 //the code will be executed on 16 August, 2001
 //every second from 11:11 to 11:12
}
```

```
OnTime("0",D,X,Y,"21","0","0")
{
 //the code will be executed every Monday
 // at 21:00:00
}
```

3. **OnEvent**(source type, number,event)—running if there is a specific event from the system object. This is the main procedure when writing scripts.
   Examples of use:

```
OnEvent("GRAY","1","ON")
{
 //will be executed when closing sensor 1
}
```

```
OnEvent("CAM","12","MD_START")
{
 //will be executed when motion detection tool of camera 12 triggers
}
```

Each procedure that has parameters can be seen in a code many times with various parameters. When an event occurs, the system will execute those of them that have the same parameters as one that has occurred.

The procedure parameter can be defined or not. If it is defined, then its value is in quotes, otherwise the parameter is written in Latin letters and the procedure will be executed for all events for which it can be defined.

Examples of use:

```
OnEvent("GRAY","1","ON") // will be executed when closing sensor 1
{
 i=1;
 i=i+1; //as variables are string, then the sum will be 11
 j=1;
 j=str(j+1); // str is a number-to-string conversion function. Inside the str
  //function all string variables (if any) are converted to integers and
  //then all integers are added together, therefore, the sum will be 2.
}
```

```
OnEvent("GRAY",N,"ON") //will be executed when any sensor is closed
{
 if(strequal(N,"3")
 {
 // will be executed if this is sensor 3
 }
}
```

Creating custom procedures

All custom procedures described in the script must be in the same program body and before procedures in which they are called.

```
procedure ProcedureName(list of parameters){
 //procedure body
}
```

> ⚠ **Attention!**
>
> The names of parameters must consist of one uppercase character.

Examples of use:

```
procedure ProcedureName(A,B)

{
 n=A+" "+B;
 //when running macro 1 n=«Macro 1», when running macro 16 n=«Macro 16»
}


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

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

Description of operators

The list of operators used to describe actions:

1. **DoReact**(object type, number, action[,Parameters])—execute an action.
   Example of use:

```
OnEvent("GRAY","1","ON")
{
 DoReact("GRELE","1","ON"); //close relay 1 when closing sensor 1
}
```

2. **DoCommand**(command line)—run the command line.
   Examples of use:

```
OnEvent("GRAY","1","ON")
{
 DoCommand("notepad.exe"); //when sensor 1 is closed, run "Notepad"
}
```

3. **Wait**(number of seconds)—wait for N seconds;
   Sleep(number of milliseconds)—wait for N milliseconds.
   Await operators must be in a single thread. Single thread must be inside square brackets.
   Example. When Sensor 1 is closed, Relay 1 is closed for five seconds.

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

4. Function to check the object state:
   CheckState(object type, number, state)—the result is 1, if the state of an object is factually accurate, otherwise 0.
   Expressions can be used as parameters. Constant values are quoted.
   Example. Check the state of camera 2 when closing sensor 1 and if the state is "Alarmed", then close relay 1

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

5. **Conditional operator**:

```
If (expression)
{
 ... // if the result is not equal to 0
}
else
{
 ... // if the result is equal to 0
}
```

   else {} part can be absent.
   Example of use:

```
OnEvent("MACRO","1","RUN"){
 x=5;
 if(x>10) {y=2;} // if "x" is greater than 10, then y=2
 else {y=3;} //otherwise y=3
}
```

6. **For operator**:

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

Expression1 is executed at the beginning of the loop; loop body is executed if expression2 is true; expression3 is executed after each execution of the loop body.
Example. When sensor1 is closed, relay1 is closed and opened every second and it will happen 10 times.

```
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**(object type, number, state)—function that creates reactions of system objects. Meanwhile, the created reaction is sent to all cores connected over the network.
Example. When running macro 1, camera 1 is armed.

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

8. **NotifyEventGlobal**(object type, number, state)—function that creates system events. Meanwhile, the created events are sent to all cores connected over the network.
Example. When running macro 1, create event “Recording” for camera 1. The command is sent to all cores as an event in order to be logged.

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

Note

If there is no need to send event to all system cores, then use the **NotifyEvent** function.

Operators and expressions

The table below lists and describes comparison, arithmetic and conditional operators.

Operator	General description, example of use
Comparison operators	
>	Comparison operator—greater. See example in Description of operators
<	Comparison operator—less. See example in Description of operators
Arithmetic operators	

+	Addition operator. Example of use:
	
	`OnEvent ("MACRO","1","RUN")`
	`{`
	` x=5;`
	` y=10;`
	` i=x+y; // add strings, i.e. 5+10=510`
	` e=str(x+y); // add integers 5+10=15`
	`}`

-	Subtraction operator. Example of use:
	
	`OnEvent ("MACRO","1","RUN")`
	
	`{`
	` x=5;`
	` y=10;`
	` i=x-y; // subtract integers 5-10=-5`
	` e=str(x-y); // subtract integers 5-10=-5`
	`}`

*	Multiplication operator. Example of use:
	
	`OnEvent ("MACRO","1","RUN")`
	`{`
	` x=5;`
	` y=10;`
	` i=x*y; // multiply integers 5*10=50`
	` e=str(x*y); // multiply integers 5*10=50`
	`}`

/	Division operator. Example of use:
	
	`OnEvent ("MACRO","1","RUN")`
	`{`
	` x=5;`
	` y=10;`
	` i=x/y; // divide integers 5/10=0.5`
	` e=str(x/y); // divide integers 5/10=0.5`
	`}`

%	Remainder after integer division. Example of use.
	`OnEvent ("MACRO","1","RUN")`
	`{`
	`a=1120.0;`
	`b=100;`
	`e=a%b; // remainder after integer division, i.e 1100 is divided by 100 and 20 is remainder.`
	`// if there is division without remainder, then result is 0`
	`}`

( )	Group of arithmetic operators. Example of use.
	`OnEvent ("MACRO","1","RUN")`
	`{`
	`x=100/((5*8)/1.028);`
	`}`

Logical operators	

&&	Logical AND operator. Example of use:
	`OnEvent ("MACRO","1","RUN")`
	`{`
	`a=1;`
	`b=2;`
	`z=3;`
	`if((a<b)&&(b<z))`
	`{`
	`y=1; //if false, then else`
	`}`
	`else`
	`{`
	`x=0;`
	`}`
	`}`

!	Logical inversion operator. Example of use:
	`OnEvent ("CAM",N,"MD_START")`
	`{`
	`if(!(strequal(N,"1",)))`
	`{`
	`DoReact("GRELE","1",""ON)`
	`}`
	`else`
	`{`
	`DoReact("GRELE","2",""ON)`
	`}`
	`}`

Description of functions

General description and examples of use of math functions, conversion functions, as well as format functions and string functions are represented in the table.

Functions **(The number of executable parameters is specified in square brackets)**	**General description, example of use**
MATH	
sin[1]	Trigonometric function for calculating the sine of an angle. Format: y=sin(x); where y—function value, x—argument of function (in radians) Example: y=sin(1.6) Event received: Event : CORE VAR_CHANGED nt_obj_id<1>,value<0.997495>,name<y>,time<15:26:41>, date<21-09-04>
cos[1]	Trigonometric function for calculating the cosine of an angle. Format: y=cos(x); where y—function value, x—argument of function (in radians) Example: y=cos(2.2) Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<-0.588501>,name<y>,time<16:00:45>, date<21-09-04>
tan[1]	Trigonometric function, returns the tangent of an angle. Format: y=tan(x); where y—function value, x—argument of function (in radians) Example: y=tan(1) Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<1.557408>,name<y>,time<16:43:45>, date<21-09-04>
asin[1]	Returns the arc sine of the specified numeric expression. Format: y=asin(x); where y—function value (in radians), x—argument Example: y=asin(0.5) Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<0.523599>,name<y>,time<16:46:39>, date<21-09-04>

acos[1]	Returns the arc cosine of the specified numeric expression. Format: y=acos(x); where y—function value (in radians), x—argument Example: y=acos(0.55) Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<0.988432>,name<y>,time<16:46:39>, date<21-09-04>
atan[1]	Returns the arc tangent of the specified numeric expression. Format: y=atan(x); where y—function value (in radians), x—argument Example: y=atan(1.2) Event received: Event : Event : CORE VAR_CHANGED int_obj_id<1>,value<0.876058>,name<y>,time<17: 07:09>,date<21-09-04>
sinh[1]	The sinh function returns hyperbolic sine of the argument value. Format: y=sinh(x); where y—function value, x—argument of function Example: y=sinh(0.8) Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<0.888106>,name<y>,time<17:12:26>, date<21-09-04>
cosh[1]	The cosh function returns hyperbolic cosine of the argument value. Format: y=cosh(x); where y—function value, x—argument of function Example: y=cosh(0.35) Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<0.336376>,name<y>,time<17:25:25>, date<21-09-04>
tanh[1]	Trigonometric function for an angle calculation. Format: y=tanh(x); where y—function value, x—argument of function Example: y=tanh(0.35) Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<1.419068>,name<y>,time<17:25:25>, date<21-09-04>

exp[1]	Returns the value of the e^x function, where x—specified numeric expression. Format: y=exp(x); where y—function value, x—argument Example: y=exp(1.65) Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<5.20698>,name<y>,time<17:39:22>, date<21-09-04>
log[1]	Returns the natural logarithm (base-e) of the specified numeric expression. Format: y=log(x); where y—function value, x—argument Example: y=log(0.65) Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<-0.430783>,name<y>,time<17:43:22>, date<21-09-04>
log10[1]	Returns the common logarithm (base-10) of the specified numeric expression. Format: y=log10(x); where y—function value, x—argument Example: y=log10(0.05) Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<-1.30103>,name<y>,time<17:46:28>, date<21-09-04>
sqrt[1]	Returns the square root of the specified numeric expression. Format: y=sqrt(x); where y—function value, x—argument Example: y=sqrt(9) Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<3>,name<y>,time<17:25:25>,date<21-09-04>
abs[1]	The abs function returns the absolute value of the argument. Format: y=abs(x); where y—function value, x—argument Example: y= abs(-1) Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<1>,name<y>,time<13:39:37>,date<22-09-04>

deg[1]	Trigonometric function for an angle calculation. Returns the grade measure. Format: y=deg(x); where y—function value in grades, x—argument value in radians Example: y=deg(3.14) Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<179.908748>,name<y>,time<13:13:51>, date<22-09-04>
rad[1]	Trigonometric function for an angle calculation. Format: y=rad(x); where y—function value in radians, x—argument value in grades Example: y=rad(180) Event received: Event : CORE VAR_CHANGED value<3.141593>,name<y>,time<15:04:17>,date<17-03-08>
CONVERSION	
floor[1]	Integer conversion function (rounding downward). Format: x= floor(y); where x—function value, y—fraction or integer Example: x= floor(5.55) Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<5>,name<x>,time<20:51:48>,date<21-09-04>
ceil[1]	Integer conversion function (rounding upward). Format: x= ceil (y); where x—function value, y—fraction or integer Example: x= ceil(5.55) Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<6>,name<x>,time<20:51:48>,date<21-09-04>

str[1]	Integer-to-string conversion function.
	
	Format: x=str(y); where x—function value, y—argument
	
	Example:
	
	z=(9);
	
	a=str(z);
	
	b=sqrt(a);
	
	Events received:
	
	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]	String-to-integer conversion function. Format: x=atof(y); where x—function value, y—argument Example: x="0"; x=str(atof(x)+10); Event received: 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>
val[1]	Integer-to-string conversion function. Format: x=val(y); where x—function value, y—argument Example: x="10"; x=str(val(x)+2); Event received: 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]	Conversion of fraction into integer (without fractional part).
	Format: x=int(y); where x—function value, y—argument (fraction for conversion)
	Example:
	y=(2.33);
	x=int(y);
	Event received:
	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>

long2time[1]	It is used to convert specified number of seconds into time. Format: x=long2time(y); where x—function value(time), y—number in seconds Format of the initial recording (argument): <MM> Format of final recording: <HH:MM:SS> Example: x=long2time(12345); Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<03:25:45>,name<x>,time<13:53:02>, date<20-09-04>

time2long[1]	Convert time into a number of seconds.
	
	Format: x=time2 long(y); where x—value in seconds, y—time in the <hours>.<minutes> format
	
	Example:
	
	y=(0.15);
	
	x=time2long(y);
	
	Event received:
	
	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>

scalar2date[1]	Convert a number of days into a date (number of days is calculated AD).
	
	Format: x= scalar2date (y); where x—value(date), y—number of days
	
	Example:
	
	y=(731500);
	
	x=scalar2date(y);
	
	Event received:
	
	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]	Convert date to number of days (number of days is calculated AD). Format: x=scalar(y); where x—numerical value (in days), y—date Recording format: <DD.MM.YYYY> Example: x=scalar("19.10.2004") Event received: 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>

convert_num[1]	Convert a number into the string.
	
	Format: x=convert_num(y); where x—string value of the number, y—convertible number
	
	Example:
	
	y=(24009921);
	
	x=convert_num(y);
	
	Event received:
	
	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<Twenty four million nine thousand nine hundred twenty-one >,name<x>,time<12:37:20>,date<23-09-04>

convert_cur[1]	Convert a number (sum of money) into the string and add dollars and cents.
	
	Format: x=convert_cur(y); where x—string value of the sum of money, y—number (sum of money)
	
	Recording format: <DD.CC>
	
	Example:
	
	y=(17999.98);
	
	x=convert_cur(y);
	
	Event received:
	
	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<Seventeen thousand nine hundred ninty-nine dollars ninty-eight cents>,name<x>,time<12:49:30>,date<23-09-04>

FORMATTING	

number_frm[2]	Formatting a number.
	
	Format: x=number_frm(y,z); where x—function value, y—initial number, z—number of figures after the decimal
	
	Example:
	
	y=(17999.09998);
	
	x=number_frm(y,3);
	
	Event received:
	
	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]	Formatting number.
	
	Format: x=int_frm(y,z); where x—value, y- operand, z—number of output digits
	
	Example:
	
	y=(17999.99);
	
	x=int_frm(y,10);
	
	Event received:
	
	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]	Formatting currency value (from '.' to '-'). Format: x=currency_std(y); where x—function value with modified format, y—number (sum of money) Example: x=currency_std(3.62); Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<3-62>,name<x>,time<13:40:01>, date<23-09-04>

IsVarExist[1]	The function that checks a specified parameter in the event. Format: y=IsVarExist(“x”); where y—value, x—parameter If the parameter exists, then “1” returns, otherwise “0”. Example: p=IsVarExist(“param0”) Event received: 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]	The function that returns the first object ID found by a specified parameter. Id=GetObjectIdByParam ("x","y","z"); where id—returned value, x—object type, y—parameter, z—parameter value Example: Id=GetObjectIdByParam("CAM","color","0"); Event received: 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 (see value<2>), if the function returns empty value (value< >), then check if the function and its parameters are written correctly
STRING	

strequal[2]	Comparing strings.
	
	Format: x= strequal(z,y); where x—value, z and y—compared strings
	
	Example:
	
	z=str(1019);
	
	y=str(1019);
	
	x=strequal(z,y);
	
	Event received:
	
	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>» (see example above)—in the received event we get «value<>»—compared strings differ, or «value<1>»—compared strings are the same

strsub[2]	Determining if there is a substring in the string.
	
	Format: x=strsub(y,z); where x—value, y—string in which the search is performed, z—substring
	
	Example 1:
	
	z=str(888123);
	
	y=str(123);
	
	x=strsub(z,y);
	
	Event received:
	
	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>
	
	Example 2:
	
	z="67hb8vc56";
	
	y="vc";
	
	x=strsub(z,y);
	
	Event received:
	
	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>" (see example 1)—index in the initial string. Starting from this index, the first occurrence of the substring in the string is detected. If the search result is negative, the function returns value<>

strempty[1]	Determining if the string is empty. Format: x=strempty(y); where x—value (1 if string is empty), y—string Example: y=""; x=strempty(y); Event received: 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> * function value value <> means that string is not empty

straleft[2]	Left alignment.
	
	Format: x=straleft(y,z); where x—aligned string, y—string, z—alignment value
	
	Example:
	
	y=str(123456789);
	
	x=straleft(y,5);
	
	Event received:
	
	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>
	
	Note. If z is larger than the number of characters in the string, then the function adds spaces to the initial string on the right until its length becomes z

strmid[3]	Get substring.
	
	Format: x=strmid(y,z,w); where x—string value, y—string, z—string position, w—substring length
	
	Example:
	
	z=(7);//position
	
	w=(9);//length
	
	x=strmid("get substring (1 - string, 2 - position, 3 - length)",z,w);
	
	y=strmid("get substring (1 - string, 2 - position, 3 - length)",17,10);
	
	Event received:
	
	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<substring>,name<x>,time<14:18:08>, date<24-09-04>
	
	Event : CORE VAR_CHANGED int_obj_id<1>,value<1 – string>,name<y>,time<14:18:08>, date<24-09-04>

strletf[2]	Get left side of string.
	
	Format: y=strleft(s,w); where y—string value, s—string, w—length (from string beginning)
	
	Example:
	
	w=(5);//length
	
	s=("Get left side of string");//string
	
	y=strleft(s,w);
	
	Event received:
	
	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< Get left *side* of *string*>,name<s>, time<14:54:31>,date<24-09-04>
	
	Event : CORE VAR_CHANGED int_obj_id<1>,value<Get>,name<y>,time<14:54:31>, date<24-09-04>

strright[2]	Get right side of string (1—string, 2—length).
	
	Format: y=strleft(s,w); where y—string value, s—string, w—length (from string end)
	
	Example:
	
	w=(6);// length
	
	s=("Get right side of string");//string
	
	y=strright(s,w);
	
	Event received:
	
	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< Get right *side* of *string*>,name<s>, time<15:10:36>,date<24-09-04>
	
	Event : CORE VAR_CHANGED int_obj_id<1>,value<strings>,name<y>,time<15:10:36>, date<24-09-04>

strnleft[2]	Get without left side of string.
	
	Format: y=strnleft(s,w); where y—string value, s—string, w—length of left side that will be cut
	
	Example:
	
	w=(6);//length
	
	s=("get without left side of string");//string
	
	y=strnleft(s,w);
	
	Event received:
	
	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<get without left side of string>,name<s>, time<15:32:38>,date<24-09-04>
	
	Event : CORE VAR_CHANGED int_obj_id<1>,value<without left side of string>,name<y>, time<15:32:38>,date<24-09-04>

srtnright[2]	Get without right side of string.
	
	Format: y=strnright(s,w); where y—string value, s—string, w—length of right side that will be cut
	
	Example:
	
	w=(6);//length
	
	s=("get without right side of string");//string
	
	y=strnright(s,w);
	
	Event received:
	
	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<get without right *s*ide of string>, name<s>,time<15:44:31>,date<24-09-04>
	
	Event : CORE VAR_CHANGED int_obj_id<1>,value< get without right side of string>, name<y>,time<15:44:31>,date<24-09-04>

get_substr[3]	Get substring (1—string, 2—substring to start with, 3—substring to end with, "\r"—end of string).
	
	Format: y=get_substr(s,w,x); where y—value(substring), s—string, w—substring to start with, x—substring to end with("\r"—end of string)
	
	Recording format: <NN.NN>
	
	Example:
	
	s=("get substring 1234567890");//string
	
	w=("to");// substring to start with
	
	x=("\r");//substring to end with, "\r"—end of string
	
	y=get_substr(s,w,x);
	
	Event received:
	
	Event : CORE VAR_CHANGED int_obj_id<1>,value<get substring 1234567890>,name<s>, time<16:34:13>,date<24-09-04>
	
	Event : CORE VAR_CHANGED int_obj_id<1>,value<sub>,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<substring 1234567890>,name<y>, time<16:34:13>,date<24-09-04>
	
	Example:
	
	s=("get substring 1234567890");//string
	
	w=("to");// substring to start with
	
	x=(1);//substring to end with, "\r"—end of string
	
	y=get_substr(s,w,x);
	
	Event received:
	
	Event : CORE VAR_CHANGED int_obj_id<1>,value<get substring 1234567890>,name<s>, time<16:36:26>,date<24-09-04>
	
	Event : CORE VAR_CHANGED int_obj_id<1>,value<sub>,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<substring >,name<y>,time<16:36:26>, date<24-09-04>

strltrim[1]	Remove spaces on the left.
	
	Format: y=strltrim(w); where y—result string value, w—string
	
	Example:
	
	w=(" remove spaces on the left");//string
	
	y=strltrim(w);
	
	Event received:
	
	Event : CORE VAR_CHANGED int_obj_id<1>,value< remove spaces on the left>,name<w>, time<17:07:49>,date<24-09-04>
	
	Event : CORE VAR_CHANGED int_obj_id<1>,value<remove spaces on the left>,name<y>, time<17:07:49>,date<24-09-04>

strrtrim[1]	Remove spaces on the right.
	
	Format: y=strrtrim(w); where y—result string value, w—string
	
	Example:
	
	w=("Remove spaces on the right ");//string
	
	y=strrtrim(w);
	
	Event received:
	
	Event : CORE VAR_CHANGED int_obj_id<1>,value<Remove spaces on the right >, name<w>,time<17:18:35>,date<24-09-04>
	
	Event : CORE VAR_CHANGED int_obj_id<1>,value<Remove spaces on the right>,name<y>, time<17:18:35>,date<24-09-04>

stratrim[1]	Remove spaces on both sides.
	
	Format: y=stratrim(w); where y—result string value, w—string
	
	Example:
	
	w=(" remove spaces on both sides ");//string
	
	y=stratrim(w);
	
	Event received:
	
	Event : CORE VAR_CHANGED int_obj_id<1>,value< remove spaces on both sides >, name<w>,time<17:27:44>,date<24-09-04>
	
	Event : CORE VAR_CHANGED int_obj_id<1>,value<remove spaces on both sides>,name<y>, time<17:27:44>,date<24-09-04>

Note

The date<DD-MM-YY> and time<HH:MM:SS> functions return the current date and time. The pi<3, 14159265358979323846264338327 95> function returns the value of π.

Examples of scripts in the embedded language

Examples with Cameras and Video surveillance monitors

GRABBER Video capture device

MACRO Macro

CAM Camera

MONITOR Monitor

Formats and functions

Format of events procedure for the **Video capture device**:

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

Operator format to describe actions with the **Video capture device**:

```
DoReact("GRABBER","_id_","_command_" [,"_parameters_"]);
```

Format of the event procedure for the **Camera** object:

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

Operator format to describe actions with the **Camera**:

```
DoReact("CAM","_id_","_command_" [,"_parameters_"]);
```

Function to check the state of the **Camera** object:

```
CheckState("CAM","number","state")
```

Format of event procedure for the **Monitor** object:

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

Operator format to describe actions with the **Monitor**:

```
DoReact("MONITOR","_id_","_command_"[,"_parameters_"]);
```

Examples

Examples of using events and reactions of the **Video capture device** object:

1. It is required to set the first channel for the first video capture device, maximum speed of digitizing, resolution is half-frame and PAL format when starting the first macro.

```
OnEvent("MACRO","1","RUN") // start macro 1
{
    DoReact("GRABBER","1", "SETUP", "chan<1>,mode<0>,resolution<1>,format<PAL>");
    //set channel 1 for the first video capture device, speed of digitizing is maximum, resolution is
half-frame, format is PAL
}
```

2. Set disks D:\ and F:\ for recording video archive when starting the third macro.

```
OnEvent("MACRO","3","RUN") //start macro 3
{
    DoReact("GRABBER","1","SET_DRIVES","drives<D:\,F:\>"); //record the video archive on disks D:\ and F:
\
}
```

3. It is required to display the first camera on the first analog output and disable the first analog outputs of the first and second cards when there is an error of connection to the second video capture device.

```
OnEvent("GRABBER","2"," UPS_FATAL_ERROR") //error of connection to the video capture device 2
{
    DoReact("CAM","1","MUX1"); //display camera 1 on the 1-st analog output of card
    Wait(5);
    DoReact("GRABBER","1","MUX1_OFF"); //disable 1-st analog output of the first card
    DoReact("GRABBER","2","MUX1_OFF"); //disable 1-st analog output of the second card
}
```

> **Note**
>
> If analog outputs of two or more cards are connected in parallel and, for example, camera 1 belongs to the first grabber and camera 2 belongs to the second grabber, then when running the «DoReact("CAM","1","MUX1");» command, it is required to run the «DoReact("GRABBER","2","MUX1_OFF");» command first. And correspondingly when running the «DoReact("CAM","2","MUX1");» command, it is required to run the «DoReact("GRABBER","1","MUX1_OFF");» command first. Otherwise signal overlaying will happen.

4. It is required to disable the second analog output of the video capture device when restoring the mains supply.

```
OnEvent("GRABBER","1","UPS_ONLINE")         //restoring the mains supply
{
    DoReact("GRABBER","1","MUX2_OFF");      //disable analog output 2
}
```

Examples of using events and reactions of the **Camera** object:

1. Switch camera to the color mode and start recording from it when arming the first camera.

```
OnEvent("CAM","1","ARM") //first camera is armed
{
    DoReact("CAM","1","SETUP","color<1>"); // set color mode of camera
    DoReact("CAM","1","REC"); //record from the first camera
}
```

2. Arm the first camera when disabling the fifth camera.

```
OnEvent("CAM","5","DETACH") // fifth camera is disabled
{
    DoReact("CAM","1","ARM"); //first camera is armed
}
```

3. Use half of resources when recording from the first camera (it means, if four cameras are connected through the first video capture device than the first camera will record with speed of 6 FPS, and other three cameras—with speed of 2-2.5 FPS) if it is in the alarm state.

```
OnEvent("CAM","1","MD_START") //first camera is in alarm state
{
    DoReact("CAM","1","SETUP","rec_priority<2>"); // use half of resources when recording
}
```

4. Set maximum compression synchronously with the fourth microphone of audio card on the first camera when recording on disk from the first camera.

```
OnEvent("CAM","1","REC") //first camera recording on disk
{
    DoReact("CAM","1", "SETUP", "compression<5>, audio_type<OLXA_LINE>, audio_id<4>"); //first camera,
maximum compression, synchronously with the forth microphone of audio card.
```

5. Start recording from the first camera with minimum quality in black and white mode when it isn't alarmed.

```
OnEvent("CAM","1","MD_STOP") // first camera stopped to be in alarm state
{
    value = 5;
    DoReact("CAM", "1", "SETUP", "compression<" + value + ">,color<0>");
    //start recording from the first camera with minimum quality in the black and white mode.
}
```

6. Start recording from the first camera in the “rollback” mode when it is disarmed.

```
OnEvent("CAM","1","DISARM")  //first camera is disarmed
{
    DoReact("CAM","1","REC","rollback<1>");  // Start recording from the first camera in the "rollback"
mode
}
```

7. Set new parameters of video signal when connecting the first camera.

```
OnEvent("CAM","1","ATTACH") //first camera is connected
{
    VIDEO_CANAL_ID = GETOBJECTPARAM("CAM","1","PARENT_ID"); // define ID of video channel to which the
first camera belongs
    DoReact("GRABBER",VIDEO_CANAL_ID,"SETUP","chan<0>,mode<0>,resolution<1>,format<pal>"); //set new
parameters of video channel
}
```

8. Start auto cruising on Camera 1 when Macro 2 is run.

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

9. There is a certain number of cameras (num). It is necessary to check the operation of motion detection on all cameras (can be used to check the performance of security sensors).
   To solve the problem, you can use the emulation of a linear character array (string), it means, the array of characters is filled in (in the example, it is the "N" character). Then, when the camera’s motion detection is triggered, the corresponding (to the camera ID) element of the array is changed (changed to "Y"). Thus, the output is a character array of "N" (the camera didn’t trigger) and "Y" (the camera triggered). The number of detections is counted and a message with the total number of cameras and the number of cameras that triggered is displayed. Start the check on Macro 1. Stop on Macro 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="All:"+str(num)+" Worked:"+str(fin);
    rez=MessageBox("",tmp,0);
}
```

10. When an alarm occurs on camera 1, captions must be overlaid on the video image from this camera. When the alarm ends, captions about the end of the alarm must be overlaid on the video image.

```
OnEvent("CAM","1","MD_START")
{
    DoReact("CAM","1","CLEAR_SUBTITLES","title_id<1>"); //delete all captions from video image
    DoReact("CAM","1","ADD_SUBTITLES","command<Camera 1 Alarm " + time + "\r>,page<BEGIN>,title_id<1>");
    //the time parameter allows you to include the time of event registration in captions
}

OnEvent("CAM","1","MD_STOP")
{
    DoReact("CAM","1","ADD_SUBTITLES","command<Camera 1 End of alarm " + time + "\r>,page<END>,
title_id<1>");
}
```

> **Note**
>
> When you use the page<BEGIN> and page<END> parameters, the corresponding fields in the captions database will be filled in, which will make it possible to search for data using the **Captions search** interface object.

Examples of using events and reactions of the **Monitor** object:

1. Play record from camera 1 on the monitor 4 with the specified date and time when running the first macro.

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

2. Switch to the mode of video archive viewing on the first camera of monitor 4 when printing the frame from the first camera and then go on 10 frames further starting from the specified date and time.

```
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. Zoom in the video image on the monitor screen if camera is in the alarm state and reset it when alarm is finished.

```
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. Display the layout number one on the monitor screen when running a macro.

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

5. Command of starting the video export from Camera 1 in the Monitor 1, starting from 24-10-14 17:10:38 and to 24-10-14 17:10:80 to the c:\aaa.avi file.
   Examples of export starting in three ways: using the IIDK (port 900 and 1030) and using script:
   a. **IIDK (port 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 (port 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. **Script** (start on Macro 1)

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

6. When macro 1 is run, enable mouse PTZ control on Camera 4 at Monitor 10. Disable it on Macro 2.

```
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. Display an active camera on an analog monitor.

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

8. Display an alarmed camera in the one-fold mode.

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

9. An alarm monitor that always displays a video from the last alarmed camera.

```
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+">");
    }
}
```

Examples with Computer and Display

SLAVE Computer

DISPLAY Display

Formats

Format of events procedure for the **Computer** object:

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

Operator format to describe actions with the **Computer** object:

```
DoReact("SLAVE","_id_","_command_" [,"_parameters_"]);
```

Format of events procedure for the **Display** object:

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

Operator format to describe actions with the **Display**:

```
DoReact("DISPLAY","_id_","_command_" [,"_parameters_"]);
```

Examples

Examples of using events and reactions of the **Computer** object:

1. Stop recording from camera 2 if there is no disk for archive recording.

   ```
   OnEvent("SLAVE","1"," NO_DISC")
   {
       DoReact("CAM","2"," REC_STOP");
   }
   ```

2. Get the archive depth of Camera 1 on Macro 1.

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

   As the result, the following string will be displayed in the debug window:

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

   Besides, the **Archive depth** event will be displayed in the Event Viewer and the archive depth in Days:Hours format will be specified in the **Additional information** field. This information is also displayed in the debug window in the **param0<>** event parameter.

Example of using events and reactions of the **Display** object:

1. Show first display on the CLIENT computer when activating the first time zone.

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

2. There are two displays, the first one shows the virtual monitor with cameras, the second one shows the Map object with the FSA sensors. When a camera alarm is triggered, Display 1 is shown, when a sensor alarm is triggered, Display 2 is shown, but only on the CLIENT computer.

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

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

Examples with Map

MAP Map

The format of the events procedure for the **Map**:

```
OnEvent("MAP", "_id_","_event_" [,"_parameters_"])
```

Operator format to describe actions with the **Map**:

```
DoReact("MAP","_id_","_command_" [,"_parameters_"]);
```

Example. Hide Camera 10 on Map 1 on Macro 10.

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

Examples with Archive and Edge storage

ARCH Backup archive

IPSTORAGE Edge storage

Formats

Format of events procedure for the **Backup archive** object:

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

The operator format for describing the actions with the **Edge storage**:

```
DoReact("IPSTORAGE","_id_","_command_" [,"_parameters_"]);
```

Examples

Example for the **Backup archive** object. Send corresponding message to all cores of the system if archiving via the Backup archive 1 isn't performed.

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

Example for the **Edge storage** object. Import archive from the edge storage of camera 45 Edge for the period from 11-01-19 16:00:55 to 11-01-19 17:00:55 on Macro 10.

```
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>");
}
```

Examples with Macros and Time zones

MACRO Macro

TIME_ZONE Time zone

Formats and functions

Format of events procedure for the **Macro** object:

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

Operator format to describe actions with the **Macros**:

```
DoReact("MACRO","_id_","_command_" [,"_parameters_"]);
```

Function to check the state of the **Macro** object:

```
CheckState ("MACRO","number","state")
```

Format of events procedure for the **Time zone** object:

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

Operator format to describe actions with the **Time zone**:

```
DoReact("MACRO","_id_","_command_" [,"_parameters_"]);
```

Function to check the state of the **Time zone** object:

```
CheckState ("TIME_ZONE","number","state")
```

Examples

Examples of using events and reaction of the **Macro** object:

1. Write the current position of camera to preset 1 when running macro 1.

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

2. Run macro 2 if camera 1 is armed.

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

3. Start and stop patrolling of a PTZ device on macros.

```
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. Example of an infinite loop and how to stop it. Start the cycle on macro 1, stop the cycle on macro 2.

```
OnEvent("MACRO","1","RUN") //when running macro 1
{
    //square brackets are needed to separate the wait statement into a separate thread
    [
    flag=1;
    for(a=1;flag<2;a=1) //loop statement
    {
        Sleep(500); //wait statement creates a pause of 500 milliseconds
        ff="!!!!!!!!!!!!!!!!!";
    }
    ]
}

OnEvent("MACRO","2","RUN") //when running macro 2
{
    flag=2;
}
```

Examples of using events and reactions of the **Time zone** object:

1. Display video image from the camera 1 on the monitor when activating the first time zone.

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

Examples with PTZ devices and Control devices

TELEMETRY PTZ device

TELEMETRY_EXT Keyboard

JOYSTICK Control device

Formats

Format of events procedure for the **PTZ device** object:

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

Operator format to describe actions with the **PTZ devices**:

```
DoReact("TELEMETRY","_id_","_command_" [,"_parameters_"]);
```

Format of events procedure for the **Keyboard** object:

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

Operator format to describe actions with the **Keyboard**:

```
DoReact("TELEMETRY_EXT","_id_","_command_" [,"_parameters_"]);
```

The format of events procedure for the **Control device** object:

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

Examples

Examples of using reactions of the **PTZ device** object:

1. Set autofocus when camera 1 is armed.

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

2. Rotate camera to the position specified in the first preset with the relay enabled.

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

3. Record the patrol route for Camera 1 corresponding to the PTZ device 1.1. The route consists of two points, such that to go from point 1 to point 2, you need to rotate the camera to the left at speed of 6 for two seconds. Patrolling must be performed at speed of 10. The time at each point of the route is 25 seconds. It is supposed that when the program is started, the camera is set to the position corresponding to the first point of the route.

```
OnEvent("MACRO","1","RUN")
{
    DoReact("TELEMETRY","1.1","PATROL_LEARN","cam<1>,preset<1>,tel_prior<1>,dwell<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>,dwell<25>,speed<10>,
flush_tour<1>");
}
```

4. There are two cameras with PTZ devices. Every 15 minutes you need to rotate cameras to preset 1 and take a screenshot. File name is current time.

```
OnTime(W,D,X,Y,H,M, "01")
{
    if(strequal(M,"0"))
    {
        name=H+"_"+M+"_"+S+".jpg";
        //Camera 1 PTZ device 1.1
        name1="Camera1 "+name;
        DoReact("TELEMETRY","1.1","GO_PRESET","preset<1>,tel_prior<1>");
        DoReact("MONITOR","1","EXPORT_FRAME","cam<1>,file<d:\"+name1);
        //Camera 2 PTZ device 1.2
        name="Camera2 "+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";
        //Camera 1 PTZ device 1.1
        name1="Camera1 "+name;
        DoReact("TELEMETRY","1.1","GO_PRESET","preset<1>,tel_prior<1>");
        DoReact("MONITOR","1","EXPORT_FRAME","cam<1>,file<d:\"+name1);
        //Camera 2 PTZ device 1.2
        name="Camera2 "+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";
        //Camera 1 PTZ device 1.1
        name1="Camera1 "+name;
        DoReact("TELEMETRY","1.1","GO_PRESET","preset<1>,tel_prior<1>");
        DoReact("MONITOR","1","EXPORT_FRAME","cam<1>,file<d:\"+name1);
        //Camera 2 PTZ device 1.2
        name="Camera2 "+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";
        //Camera 1 PTZ device 1.1
        name1="Camera1 "+name;
        DoReact("TELEMETRY","1.1","GO_PRESET","preset<1>,tel_prior<1>");
        DoReact("MONITOR","1","EXPORT_FRAME","cam<1>,file<d:\"+name1);
        //Camera 2 PTZ device 1.2
        name="Camera2 "+name;
        DoReact("TELEMETRY","1.2","GO_PRESET","preset<1>,tel_prior<1>");
        DoReact("MONITOR","1","EXPORT_FRAME","cam<2>,file<d:\"+name);
    }
}
```

5. Patrol multiple FOVs using the PTZ camera presets, with the possibility of activating the motion detection on certain areas. Camera 1: five detection areas, five presets. These two parameters are set by the n variable. Macro 1 starts the algorithm. Macro 2 stops the algorithm. Flag is an internal variable.
   When the algorithm starts, the camera sets into preset 1 and arms detection area 1. There is a delay of 200 milliseconds between these commands, so that the camera has time to set into the preset. Then after five seconds, area 1 is disarmed, and the cycle starts again, but with area 2 and preset 2. And so on until all n areas and presets are run through. After that, the algorithm starts again from 1. The algorithm stops if the flag variable is reset (using macro 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;
}
```

Example of using events and reactions of the **Keyboard** object:

Turn on the light and arm camera 2 after pressing the key 15 on the *AXIS T8312* keyboard.

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

Example with Core

CORE

Procedure is started when the corresponding event occurs. Format of events procedure for the **Core** object:

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

Example. When a face appears in the frame, display the video image from the corresponding camera on Monitor 2. When the face disappears, remove the video image from the corresponding camera from Monitor 2.

```
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+">");
        ]
    }
}
```

Examples with Incident server and Incident manager

INC_MANAGER Incident manager

INC_SERVER Incident server

Format of event procedure for the **Incident manager** object:

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

The format of event procedure for the **Incident server** object:

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

The operator format for describing the actions with the **Incident server** object:

```
DoReact("INC_SERVER","_id_","_command_" [,"_parameters_"]);
```

Examples with Operator protocol and Event Viewer

OPERATORPROTOCOL Operator protocol

EVENT_VIEWER Event Viewer

Formats

Format of events procedure for the **Operator protocol** object:

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

Operator format for describing the actions with the **Operator protocol**:

```
DoReact("OPERATORPROTOCOL","_id_","_command_" [,"_parameters_"]);
```

Format of events procedure for the **Event Viewer** object:

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

Operator format for describing the actions with the **Event Viewer**:

```
DoReact("EVENT_VIEWER","_id_","_command_" [,"_parameters_"]);
```

Examples

Examples of using events and reactions of the **Operator protocol** object:

1. Delete the first alarm on Camera 3 from the Operator protocol 1 window on Macro 2.

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

2. Hide the Alarm situation, Suspicious situation and False alarm buttons for the Disarm event from Camera 12 in the Operator protocol 1 window on Macro 2.

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

Example of using events and reactions of the **Event Viewer** object:

Set general background color to black and general text color to white for Event Viewer 1 on Macro 1.

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

Examples with Operator query panel and SIP-terminal

DIALOG Operator query panel

SIP_TERMINAL SIP-terminal

Formats

Operator format to describe actions with the **Operator query panel**:

```
DoReact("DIALOG","_id_","_command_" [,"_parameters_"]);
```

Format of events procedure for the **SIP-terminal** object:

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

Operator format to describe actions with the **SIP-terminal**:

```
DoReact("SIP_TERMINAL","_id_","_command_" [,"_parameters_"]);
```

Examples

Examples of using reactions of the **Operator query panel** object:

1. Using macro 1, set coordinates of the left top corner of the operator query panel (PANASONIC-850 PTZ camera) in the center of the screen, prohibit its moving and display it on the screen.

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

2. Close the operator query panel on macro 2.

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

Examples with Audio

PLAYER Audio player

OLXA_LINE Microphone

Formats

Operator format to describe actions with the **Audio player**:

```
DoReact("PLAYER","_id_","_command_" [,"_parameters_"]);
```

Format of events procedure for the **Microphone**:

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

Operator format to describe actions with the **Microphone**:

```
DoReact("OLXA_LINE ","_id_","_command_" [,"_parameters_"]);
```

Function to check the state of the **Microphone** object:

```
CheckState("OLXA_LINE","number","state")
```

Examples

Examples of using events and reactions of the **Audio player** object:

1. Playback the audio file when the camera stops recording:

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

2. Stop playing back the audio file when the camera starts recording:

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

3. Playback the audio file from the occurrence of an event to the occurrence of another event (in this example, it is the start of macros).
   Audio file must last no longer than the number of seconds specified in the Wait statement.

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

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

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

Examples of using events and reactions of the **Microphone** object:

1. Turn on the first microphone when the sound activated recording is enabled.

```
OnEvent("OLXA_LINE","1","accu_start") //enable sound activated recording
{
    DoReact("OLXA_LINE","1","ARM"); //enable record from microphone
}
```

2. Set minimum compression on microphone when disabling record of audio signal.

```
OnEvent("OLXA_LINE","1","DISARM") // disable record from microphone
{
    DoReact("OLXA_LINE","1","SETUP","compression<5>"); //minimum compression is set up
}
```

3. The audio from the microphone (OLXA_LINE) isn't recorded synchronously with the camera. By default, the microphone isn't armed. It is necessary to record audio both on sound activation and on camera detection. When sound activated recording (ACCU_START) and motion detection start, forced audio recording is enabled, and the flag variable is incremented by one. At the end of sound activated recording and motion detection, the flag variable is decremented by one, and audio recording stops only if it is equal to zero, it means, there is neither sound activation nor motion.

```
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");
    }
}
```

Example with Videogate

GATE Videogate

Format of events procedure for the **Videogate** object:

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

Operator format for actions with the **Videogate**:

```
DoReact("GATE","id","command"[,"parameters"]);
```

Example. Send corresponding messages to all system cores when the input speed on the gate 1 is reduced.

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

Examples with Detection

CAM_VMDA_DETECTOR VMDA detection

CAM_FACECAPTURE Face Detection

CAM_IP_DETECTOR Embedded detection

Formats

Format of events procedure for the **VMDA Detection**:

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

Operator format to describe actions with the **VMDA Detection**:

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

Format of events procedure for the **Face Detection** object:

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

Operator format to describe actions with the **Face Detection**:

```
DoReact("CAM_FACECAPTURE","_id_","_command_" [,"_parameters_"]);
```

Format of events procedure for the **Embedded detection** object:

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

Example

Example of using events and reactions of the **VMDA Detection** object:

Arm the VMDA Detection 2 on Macro 1:

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

Example with User

PERSON User

Format of the events procedure for the **User** object:

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

Examples with Captions

TITLEVIEWER Captions search

CAM_TITLE Captioner

Formats

Format of events procedure for the **Captions search** object:

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

Operator format to describe actions with the **Captioner**:

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

Examples

Example for the **Captions search** object:

When double-clicking the search result line in the Captions search window, display the video archive corresponding to this result on the monitor 4.

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

Example for the **Captioner**.

Start the update of the captions database on Macro 1.

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

Examples with System restart service and Failover service

SSS_WATCHDOG System restart service

FAILOVER Failover service

Formats

Format of events procedure for the **System restart service** object:

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

Operator format to describe actions with the **System restart service**:

```
DoReact("SSS_WATCHDOG","_id_","_command_" [,"_parameters_"]);
```

Format of events procedure for the **Failover service** object:

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

Operator format to describe actions with the **Failover service**:

```
DoReact("FAILOVER","_id_","_command_" [,"_parameters_"]);
```

Example

Examples of using events and reactions of the **System restart service** object:

Activate the third camera on monitor 5 when restarting the module.

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

Example with BacNet

BACNET BacNet

Format of events procedure for the **BacNet** object:

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

Operator format to describe actions with the **BacNet** object :

```
DoReact("BACNET","_id_","_command_" [,"_parameters_"]);
```

Examples with Relay and Sensors

GRELE Relay

GRAY Sensor

Formats and functions

Format of events procedure for the **Relay**:

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

Operator format to describe actions with the **Relay**:

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

Function to check the state of the **Relay** object:

```
CheckState("GRELE","number","state")
```

Format of events procedure for the **Sensor**:

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

Operator format to describe actions with the **Sensor**:

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

Function to check the state of the **Sensor** object:

```
CheckState ("GRAY","number","state")
```

Examples

Example of using events and reactions of the **Relay** object:

Enable relay 2 when connection with relay 1 is lost.

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

Examples of using events and reactions of the **Sensor** object:

1. It is required to switch the second sensor over to the second input if connection with the first sensor is lost.

   ```
   OnEvent("GRAY","1"," SIGNAL_LOST") //connection with first sensor is lost
   {
       DoReact("GRAY","2","SETUP","chan<2>"); //sensor is on the second input
   }
   ```

2. Open the second sensor and enable the rollback record of the first camera when the first sensor is closed.

```
OnEvent("GRAY","1"," ON") //first sensor is closed
{
    DoReact("GRAY","2","SETUP","type<1>"); //open the second sensor
    DoReact("CAM","1","REC","rollback<1>");//perform rollback record from the first camera
}
```

Examples with Message services and notification services

Formats

Format of events procedure for the **Mail Message Service**:

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

Operator format to describe actions with the **Mail Message Service**:

```
DoReact("MMS","_id_","_command_" [,"_parameters_"]);
```

Format of events procedure for the **Mail message**:

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

Operator format to describe actions with the **Mail message**:

```
DoReact("MAIL_MESSAGE","_id_","_command_" [,"_parameters_"]);
```

Operator format to describe actions with the **Voice Message Service**:

```
DoReact("VMS","_id_","_command_" [,"_parameters_"]);
```

Operator format to describe actions with the **Voice Notification Service**:

```
DoReact("VNS","_id_","_command_" [,"_parameters_"]);
```

Format of events procedure for the **Short Message Service** object:

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

Operator format to describe actions with the **Short Message Service**:

```
DoReact("SMS","_id_","_command_" [,"_parameters_"]);
```

Format of events procedure for the **Telegram bot**:

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

Operator format to describe actions with the **Telegram bot**:

```
DoReact("TELEGRAM","_id_","_command_" [,"_parameters_"]);
```

Examples

Example of using reactions of the **Mail Message Service** object.

Set port number of the mail message service to 25 on Macro 1.

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

Example of using reactions of the **Mail message** object.

Send message with image from camera when it switches to an alarm state when motion detection triggers.

```
OnInit(){
    i=0; //counter is used to avoid overwriting of images from one camera
}

OnEvent("CAM",N,"REC") //camera is in alarm state

{
    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<camera is triggered"+ N + ">, subject<alarm on camera>,
from<john.smith@axxonsoft.com>, to<mary.foreman@axxonsoft.com>,attachments<" + filename + ">");
    DoReact("MAIL_MESSAGE","1","SEND");
}
```

Example of using reactions of the **Voice Message Service** object:

It is required to send message on macro 1 if modem is connected to COM2 port, type of dialing is pulse, don't wait for tonal signal.

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

Examples of using events and reactions of the **Voice Notification Service** object:

1. Play the audio file when camera stops recording:

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

2. Stop playing the audio file when camera starts recording:

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

3. When a predetermined time zone starts, change the volume control value to a lower one, and then after the time zone ends, set the average volume control value:

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

Examples of using events and reactions of the **Short Message service** object:

1. It is required to send a short message to the "89179190909" number when the first camera is alarmed.

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

2. Install a short message device and send a message to the "89179190909" number when the first sensor is alarmed.

```
OnEvent("GRAY","1","CONFIRM") //confirm alarm from sensor 1
{
    DoReact("SMS","1","SETUP","device<>",); //install a device for short message delivery
    DoReact("SMS","1","SETUP","phone<+79179190909>,message<sensor 1, alarm>"); //send message about an
alarm on sensor 1 to telephone number
}
```

3. Play the c:\Windows\Media\Tada.wav audio file when receiving an sms using the Mail Message Service 2.

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

Examples of calling the command for sending a message to Telegram using a macro:

```
OnEvent("MACRO","3","RUN") //run macro 3
{
  //Sending using chat_id & bot_id from object settings:
  DoReact("TELEGRAM",1,"SEND","text<Hello world>");

  //Explicit specifying of chat_id & bot_id in the command:
  DoReact("TELEGRAM",1,"SEND","text<Hello world>,chat_id<828752651>,bot_id<809045046:
AAGtKxtDWu5teRGKW_Li8wFBQuJ-l4A9h38>");

  //Sending a file with a chat ID and bot ID:
  DoReact("TELEGRAM",1,"SENDPHOTO","caption<Hello world>,chat_id<828752651>,bot_id<809045046:
AAGtKxtDWu5teRGKW_Li8wFBQuJ-l4A9h38>,photo<G:\\1.jpg>");

  //Sending geolocation with a chat ID and bot ID:
  DoReact("TELEGRAM",1,"SEND","text<Hello world>,chat_id<828752651>,bot_id<809045046:
AAGtKxtDWu5teRGKW_Li8wFBQuJ-l4A9h38>",longtitude<37.3428359>,latitude<55.6841654>,address<Office>);
}
```

Appendix 1. Priorities of the start and stop recording commands

Start and stop recording commands can have different priorities in *Axxon PSIM*. The priority of start/stop recording commands is set by the priority<> parameter of the REC and REC_STOP reactions, respectively. If you try to stop recording using the command with the priority that is lower than one of the command that initiated recording, then this command will be ignored.

When recording is started/stopped manually or by macro or by the detection tool triggering, the priority is not set. The table shows the behavior of *Axxon PSIM* when various ways to start/stop recording are in use.

Start/stop recording 1	**Start/stop recording 2**	**Behavior**
Start/stop recording is initiated by the operator using the camera context menu (start /stop recording) or by macro	Start recording at CAM|1|REC| reaction, stop recording at CAM|1|REC_STOP| reaction	The start/stop recording commands 1 and 2 are equal*
Start/stop recording is initiated by the operator using the camera context menu (start /stop recording) or by macro	Start recording at CAM|1|REC|priority<0> reaction, stop recording at CAM|1|REC_STOP|priority<0> reaction	The stop recording command 1 stops recording started using command 2
Start/stop recording is initiated by the operator using the camera context menu (start /stop recording) or by macro	Start recording at CAM|1|REC|priority<1> reaction, stop recording at CAM|1|REC_STOP|priority<1> reaction	The stop recording command 1 stops recording started using command 2
Start/stop recording is initiated by the operator using the camera context menu (start /stop recording) or by macro	Start recording at CAM|1|REC|priority<2> reaction, stop recording at CAM|1|REC_STOP|priority<2> reaction	The start/stop recording commands 1 and 2 are equal*
Start/stop recording is initiated by the operator using the camera context menu (start /stop recording) or by macro	Start/stop recording is initiated by the detection tool (for example, the main motion detection tool)	The stop recording command 1 stops recording started using command 2
Start recording at CAM|1|REC|priority<0> reaction, stop recording at CAM|1|REC_STOP|priority<0> reaction	Start/stop recording is initiated by the detection tool (for example, the main motion detection tool)	The stop recording command 2 stops recording started using command 1
Start recording at CAM|1|REC|priority<1> reaction, stop recording at CAM|1|REC_STOP|priority<1> reaction	Start/stop recording is initiated by the detection tool (for example, the main motion detection tool)	The following variants are possible: 1. If a camera is armed, the CAM|1|REC|priority<1> command starts recording and the alarm on the camera begins, then recording continues after the alarm has ended. The CAM|1|REC_STOP|priority<1> command stops recording. 2. If a camera is armed, an alarm is initiated on the camera and the CAM|1|REC|priority<1> command is sent, then recording continues after the alarm has ended. The CAM|1|REC_STOP|priority<1> command stops recording. 3. If a camera is armed, an alarm is initiated on the camera and the CAM|1|REC_STOP|priority<1> command is sent, then recording continues and it stops when the alarm has ended 4. If a camera is armed and the CAM|1|REC|priority<1> command is sent, the recording is started. If an alarm is initiated and the CAM|1|REC_STOP|priority<1> command is sent, then recording continues

Start recording at CAM|1|REC|priority<2> reaction, stop recording at CAM|1|REC_STOP|priority<2> reaction	Start/stop recording is initiated by the detection tool (for example, the main motion detection tool)	The stop recording command 1 stops recording started using command 2

* Equivalence of ways means that recording can be stopped using way 1 if it was started using way 2 and vice versa if the recording is started using way 1, then it is possible to stop recording using way 2.

Appendix 2. Defining the param_id and param_value values for the SET_IPINT_PARAM reaction

The values of the **param_id** and **param_value** parameters, required for the SET_IPINT_PARAM reaction, can be individual both for each of integrated IP cameras and for their firmwares.

The values of the **param_id** and **param_value** are defined as follows:

1. Open the directory with installed DriverPack, by default, **C:\Program Files\Common Files\AxxonSoft\Ipint.DriverPack\3.0.0**.
2. Using any word processor, open a file in this directory with the **Ipint.<Name of camera driver>.rep** name, for example, Ipint.SonyIpela.rep.

   > **Note**
   >
   > In most cases, the name of the driver is the same as the name of the manufacturer of the IP device. Contact AxxonSoft support to check the name of the driver for the required manufacturer.

3. In the file, find the name of the required model, for example, 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. There is the **<videoSourceRef>** tag within the <device> tag that contains the description of the required model like in the <model> tag. You must find one more occurrence of the **id** value of this parameter in the file (in this example this is video_source_dh160 value) in the **videoSource** tag.

```
<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. The parameters of IP device and their possible values are described in the **<property>** tags. The description of possible values depends on their type.

In this example the **param_id**="daynight" parameter can be used to switch the **Day/Night** mode on the camera. In this case the possible values of the **param_value** parameter are: auto, on, off, timer or sensor.

Example

Example of using the SET_IPINT_PARAM reaction:

1. For the **Camera** object:
   DoReact("CAM", "1","SET_IPINT_PARAM","param_id<daynight>,param_value<on>");
2. For the **Video capture device** object:
   DoReact("GRABBER", "1","SET_IPINT_PARAM","param_id<daynight>,param_value<on>,cam_id<1>");

As a result of reactions execution the value of the "daynight" parameter is "on" for Camera 1.

To enable the SET_IPINT_PARAM reaction, the multistream mode must be enabled in *Axxon PSIM (*see Configuration of multistream video). Keep in mind that if only one stream is integrated for the camera, then there will be no video in the multistream mode.

You can find out the number of integrated streams in the list of IP devices integrated with *Axxon PSIM* (see Documentation Drivers Pack).

If this way can't be used for any reason, then find out the number of integrated streams as follows:

1. Repeat steps 1–3 of the previous algorithm.

```
    <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>
```

2. The required model is described within the <device> tag, integrated video streams are described in the **<videoStreamingRef >** tags. There must be more than one stream.

The Script object. Programming using the JScript language

Purpose and features of the JScript language

Programming in JScript

- The Script system object
- The Editor-Debugger utility
- The Debug window
  - Enabling the Debug window
  - Working with Debug window
    - Copying information on event or reaction to the clipboard
    - Highlighting messages
    - Events and reactions filter
    - Searching for events and reactions
    - Clearing the Debug window
- Getting the list of system names of objects, reactions and events in Axxon PSIM

Creating your first script

Working with script

- Creating a script
- Saving a script
- Deleting a script
- Searching text in script
- Replacing text in script

Script debugging

- Script debugging features
- Creating and using test events
- Working with the debugging windows of the Editor-Debugger utility
  - Viewing the script messages
  - Displaying messages about starting, verifying, changing and executing scripts in the debugging windows
- Using third-party debugger programs

Examples of scripts in the JScript language

- Examples of scripts with Video surveillance monitor and Cameras
- Examples of scripts with Map
- Examples of scripts with detection tools
- Examples of scripts with Macros
- Example of script with Users
- Examples of scripts with Incident server and Incident manager
- Example of script with Failover service
- Examples of scripts with BacNet
- Example with Telegram bot
- Examples of scripts with Event Viewer

Appendix 1. Description of the Editor-Debugger utility

- The purpose of the Editor-Debugger utility
- The interface of the Editor-Debugger utility
  - The Editor-Debugger interface
  - The Debug-edit script tab
  - The Script messages tab
  - Main menu
    - Description of the main menu interface
    - Description of the File item of the main menu
    - Description of the View item of the main menu
    - Description of the Debug and edit item of the main menu
    - Description of the List of events item of the main menu
  - Description of the Filter dialog window
  - Description of the Highlight dialog window

Appendix 2. Creating custom objects with ability to set events, reactions and states

Export to PDF

Purpose and features of the JScript language

The JScript programming language is used in *Axxon PSIM* to implement additional user functions not included in the basic *Axxon PSIM* f unctionality.

The JScript programming language is a standard feature for developing user scripts. *Axxon PSIM* supports the version of the JScript language implemented in the ActiveX technology by Microsoft. The general description of the JScript object model, used in *Axxon PSIM*, is given in the Microsoft documentation (for example, MSDN).

The JScript scripts in *Axxon PSIM* are executed using the standard ActiveX software components included in the Windows OS. So, when you develop scripts, you can use any components of the JScript object model implemented in the ActiveX technology.

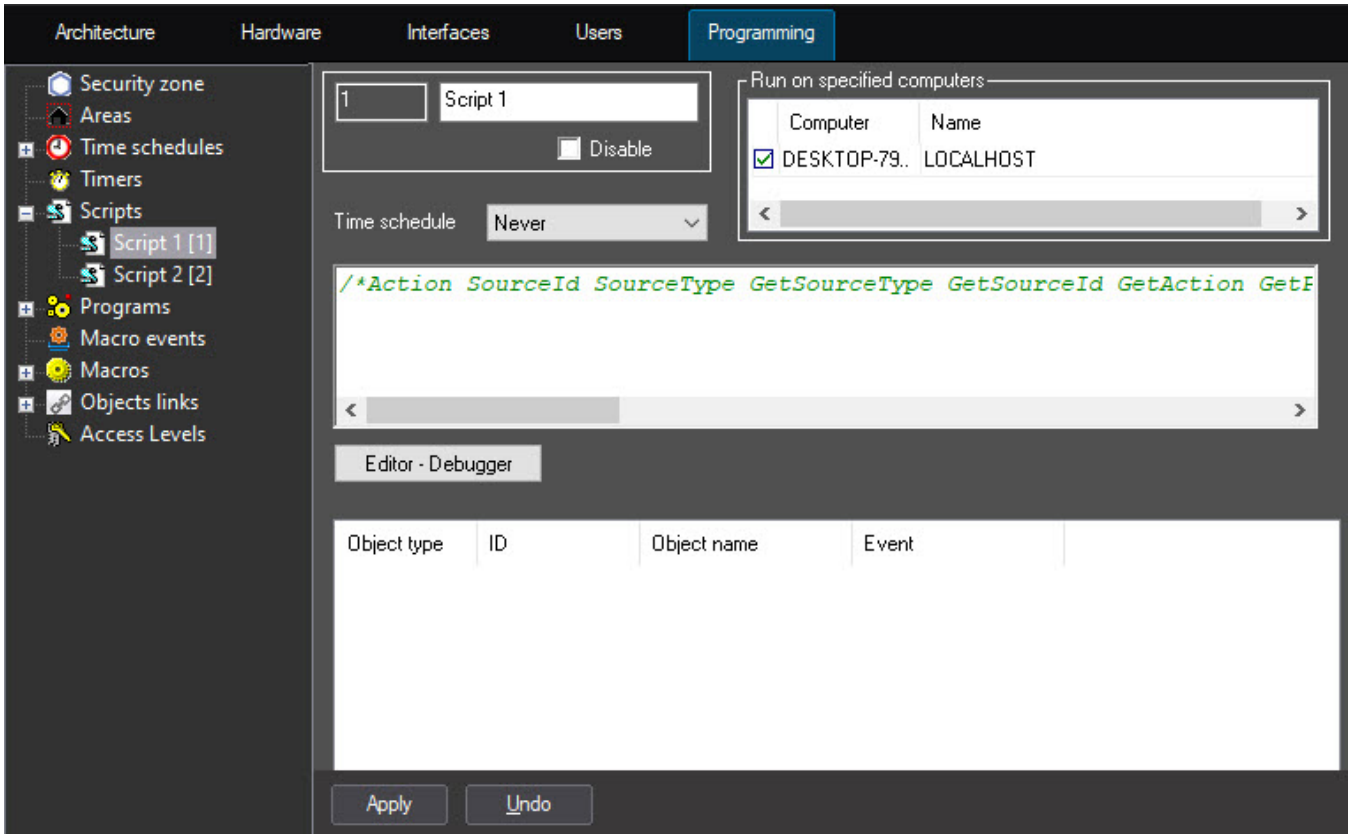
Axxon PSIM additionally provides a specialized object model for script development in the JScript language that allows you to work with system objects of *Axxon PSIM*, receive and send system events and reactions.

Programming in JScript

The Script system object

The **Script** system object is used to initialize a script developed in the JScript language in *Axxon PSIM* and to set the parameters for its execution..

The settings panel of the **Script** system object is shown in the figure below:



Attention!

Creating of large number of the **Script** objects (more than 100) can cause system instability.

The settings panel of the **Script** system object allows choosing the time schedules and the computers (kernels) for executing the script.

Note

To set all checkboxes next to all computers, select a cell in the column with checkboxes and press Ctrl+A. To clear all checkboxes, select a cell and press Shift+A.

The settings panel of the **Script** system object has the button for starting the *Editor-Debugger* utility and the text window for viewing the script text created using this utility. You can edit the script in the *Editor-Debugger* utility or directly on the settings panel for the **Sc ript** object.

Moreover, you can configure the events filter—the list of events that the **Script** system object will process. Including the event to the filter equals to the *if* operator in the text of script, it means, when the event is in the table, the operator can be omitted.

Attention!

You must configure the events filter when you create a script in large distributed configurations. Otherwise, the module will process all incoming events and it will lead to module malfunctioning.

Example.

If the **Object's type** column has the **Macro** value, the **Identifier** column has the **1** value and the **Event** column has the **Executed** value, then instead of the script below

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

you can use the following script:

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

The detailed information on the elements of the settings panel for the **Script** object is given in Administrator's guide.

Example.

Recording on Camera 1 must be started when controlling the PTZ camera in the Video surveillance monitor.

For this, configure the **Script** object as follows:

1. Select the required time schedule when the script must be executed.
2. Enter the script text:

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

3. Configure the filter as follows:
   a. From the **Object's type** drop-down list, select CORE.
   b. In the **Event** field, enter DO_REACT.

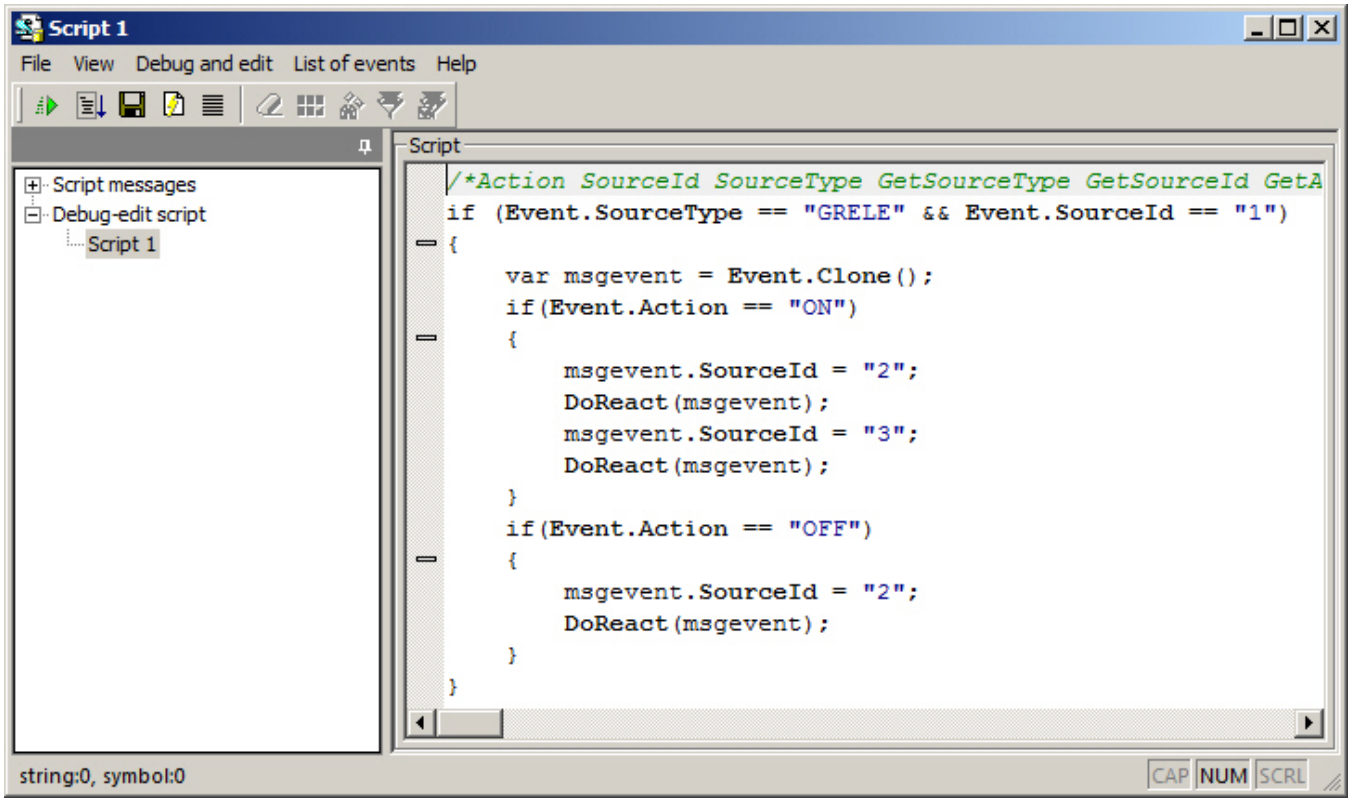
The filter can be set by the UPDATE_OBJECT event of the CORE object. Example of the command to add the **Camera** object with identifier 1 to the filter of the **Script** object with identifier 2:

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

The Editor-Debugger utility

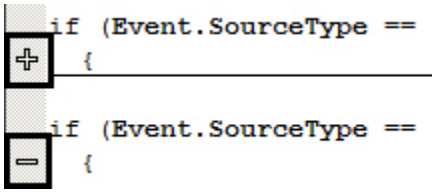
The *Editor-Debugger* utility is used to create, debug and edit scripts in *Axxon PSIM*.

The *Editor-Debugger* dialog window is shown in the figure below.



The *Editor-Debugger* utility contains the embedded text editor and the debugging window.

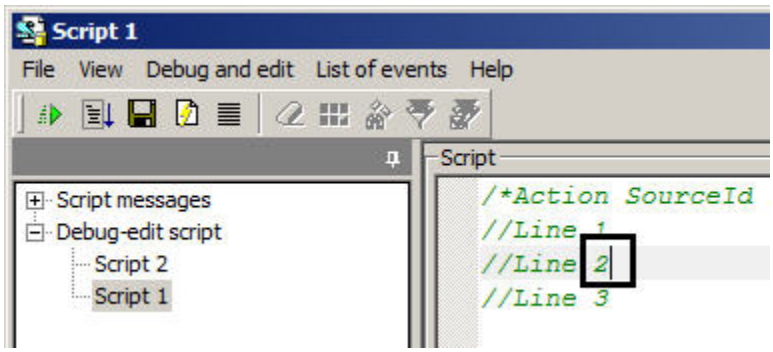
To help with writing correct scripts, the text editor automatically highlights objects, methods and properties that are a part of the object model of the JScript language. In addition, the code blocks can be collapsed or expanded with **-** or **+** buttons to the left of the **Sc ript** text field.



Note

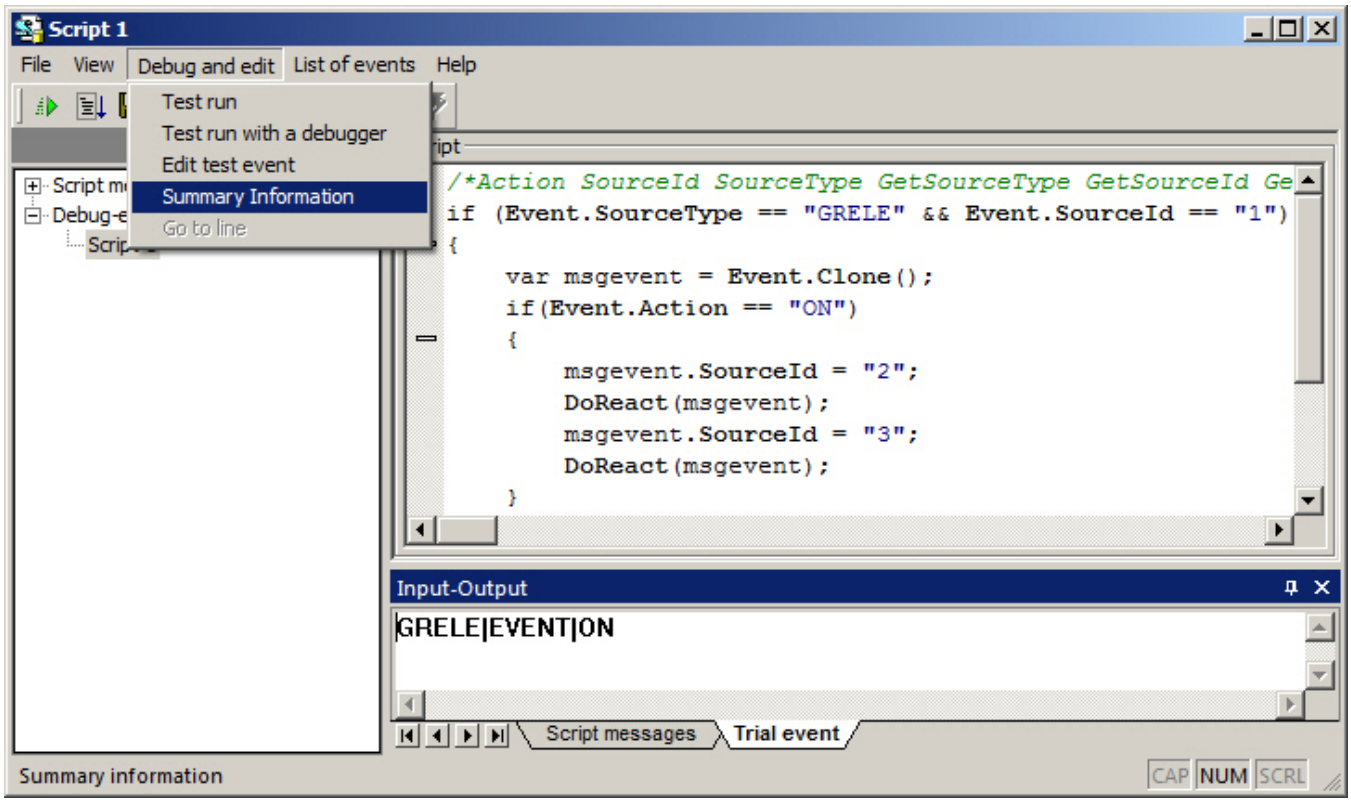
When switching between scripts, script messages or other objects in the *Editor-Debugger* utility tree, the collapsing settings are reset, it means, all blocks in the script become expanded.

The position of the cursor in each script is stored within one *Axxon PSIM* session (the cursor position is reset at restart). For example, if you place the cursor at the end of **//Line 2** in **Script 1**, then switch to **Script 2** and perform any action in it, the cursor will be still at the end of **//Line 2** when you return to **Script 1**.



The debugging window of the *Editor-Debugger* utility allows viewing the information about all events registered by the system. You can filter the events that are displayed in the debugging window. A separate debugging window is created for each **Script** system object, which allows you to debug each script individually using the filters.

To debug the script, it is possible to test run using a user-defined test event generated by the utility and not registered in the system. To display or edit this event, select **Debug and edit Summary Information**, and then go to the **Trial event** tab on the panel that opens at the bottom of the window. For details, see Creating and using test events.



You can save the created script in the **Script** system object or in a text file on the hard drive.

The Debug window

Axxon PSIM allows viewing all events and reactions happening in the system in real time. Events and reactions with object properties are displayed in the **Debug window**. You can copy them to the Windows clipboard and then use in programs.

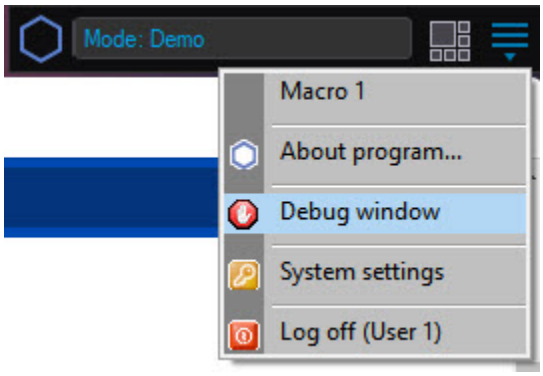
Enabling the Debug window

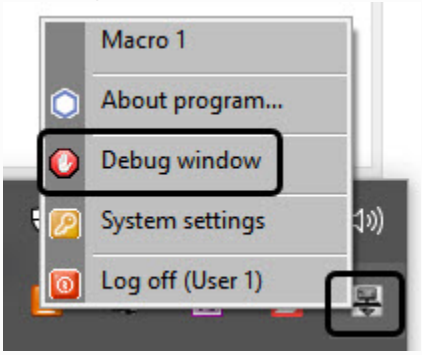
Starting with weekly build 1.0.1.172, the **Debug 4** mode is enabled by default. In previous versions, the **Debug window** is disabled by default. To enable the **Debug window**, do the following:

1. Shut down *Axxon PSIM.*
2. Run the **Advanced** *Tweaki.exe* utility.

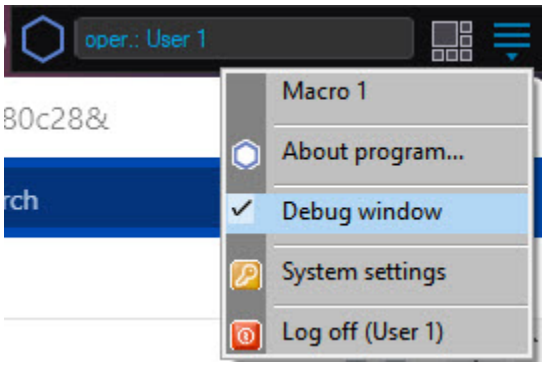
> **Note**
>
> You can enable the Debug window without using the tweaki.exe utility. For this, set values 1, 2, 3, or 4 for the Debug string parameter in the HKEY_LOCAL_MACHINE\SOFTWARE\AxxonSoft\PSIM section of the Windows registry (HKEY_LOCAL_MACHINE \Software\Wow6432Node\AxxonSoft\PSIM for 64-bit system).

3. Select **Axxon PSIM** section in the tree on the left side of the utility dialog box.
4. Change the value of the **Debug mode** parameter from **Disabled** to **Debug 1**, **Debug 2**, **Debug 3**, or **Debug 4**. Any of these modes will enable the **Debug window**, the difference between them is in the amount of information written to the log files (see The Settings panel of the Axxon PSIM section).
5. Click the **OK** button.
6. Start *Axxon PSIM.*
7. A new **Debug window** item will appear on the Main control panel of *Axxon PSIM*.



> **Note**
>
> This menu is also available in the Windows notification area (system tray)—left click the  button or short click the F8 hot key.
>
> 
>

8. Select the **Debug window** item on the Main control panel in order to display the Debug window on the screen. The selected **Debug window** item is marked with a flag.



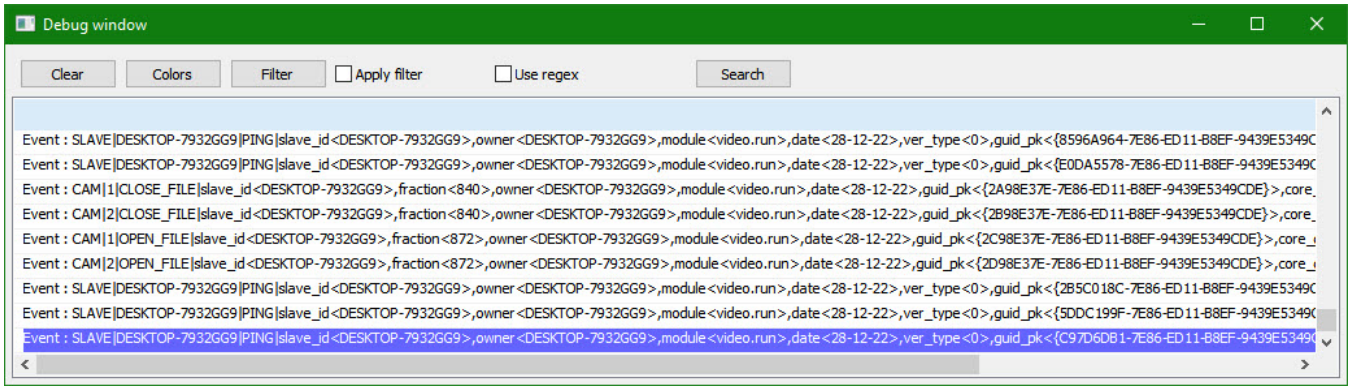
To hide the Debug window, re-select the Debug window item on the Main control panel.

Note

To disable the Debug window, select the **Disable** value for the **Debug mode** in the tweaki.exe utility or set value 0 for the Debug string parameter in the HKEY_LOCAL_MACHINE\SOFTWARE\AxxonSoft\PSIM section of the Windows registry (HKEY_LOCAL_MACHINE \Software\Wow6432Node\AxxonSoft\PSIM for 64-bit system). These actions are performed when you exited *Axxon PSIM*.

Working with Debug window

The appearance of the Debug window is shown in the figure. The Debug window displays the sequence of events and reactions in the system.



The Debug window has the following features:

1. always on top of other windows;
2. you can change the size of the Debug window using the mouse;
3. you can copy information on event or reaction to the Windows clipboard and then use it in programs;
4. you can filter events and reactions in the Debug window;
5. you can highlight events and reactions in the Debug window;
6. you can search for events or reactions in the Debug window.

You can use regular expressions to highlight and filter messages in the Debug window.

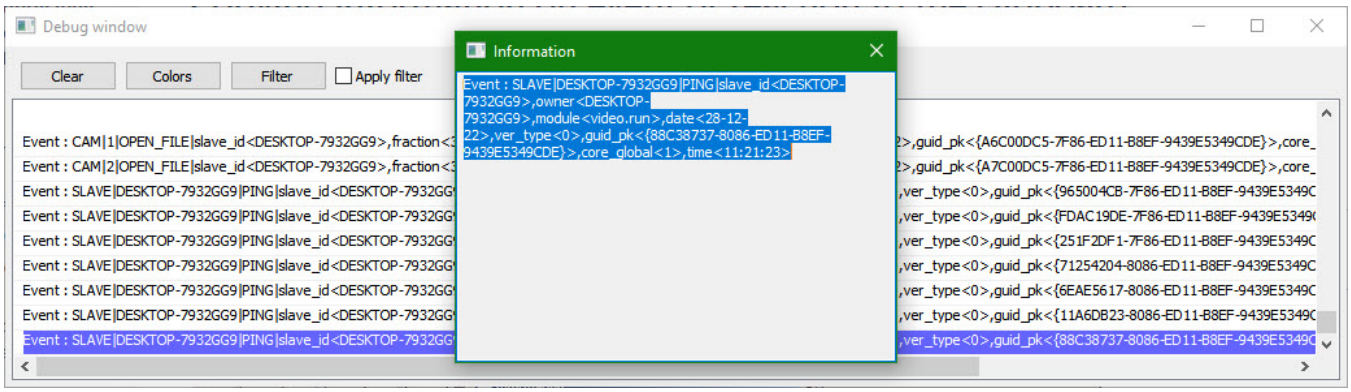
Copying information on event or reaction to the clipboard

To read and/or copy information on an event or a reaction to the Windows clipboard, do the following:

1. Select the required line in the **Debug window**.
2. Right-click the selected line. The **Information** window with the information on the required event or reaction will open.
3. Select the information that you want to copy to the Windows clipboard and press **Ctrl**+**C**.

> **Note**
>
> Use the context menu for operations with text in the **Information** window (right-click the selected text).

4. To close the **Information** window, click the  button.

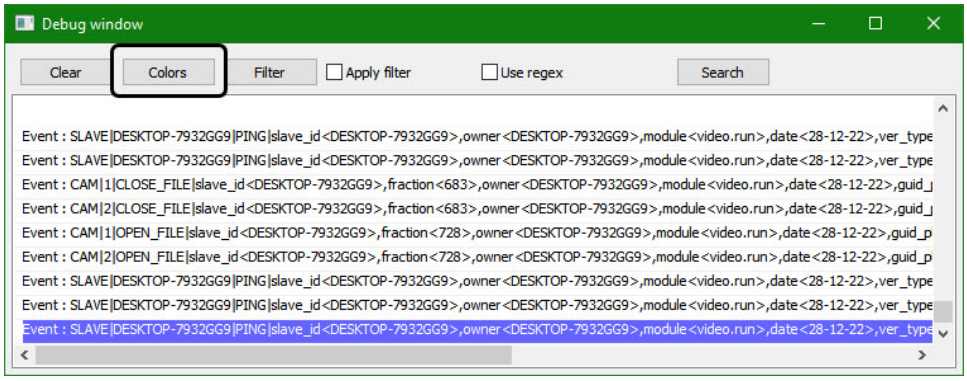


Information on event or reaction is now copied to the Windows clipboard.

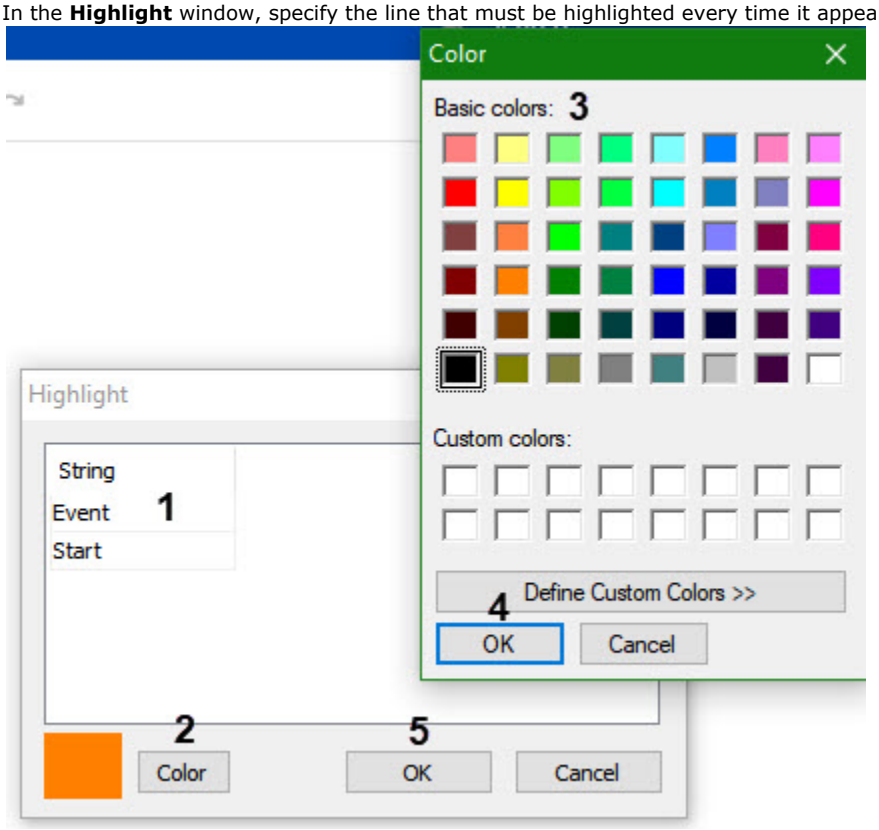
Highlighting messages

To configure message highlighting in the Debug window, do the following:

1. Click the **Colors** button.



2. In the **Highlight** window, specify the line that must be highlighted every time it appears in the message (**1**).

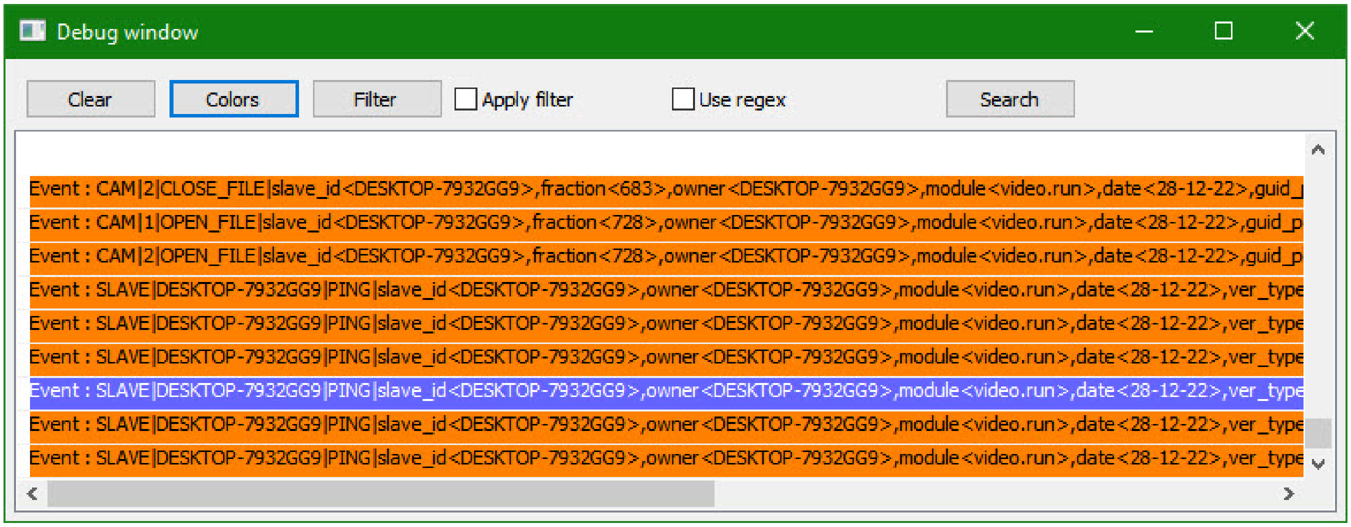


3. Click the **Color** button (**2**).
4. Select the color in the **Color** dialog window (**3**).
5. Click the **OK** button (**4**).
6. Repeat steps 2-5 for all required lines.

> **Note**
>
> To add a line to the table, press the key on the keyboard.

7. Click the **OK** button (**5**).

As a result, messages with the enterd line will be highlighted in the specified color in the Debug window.



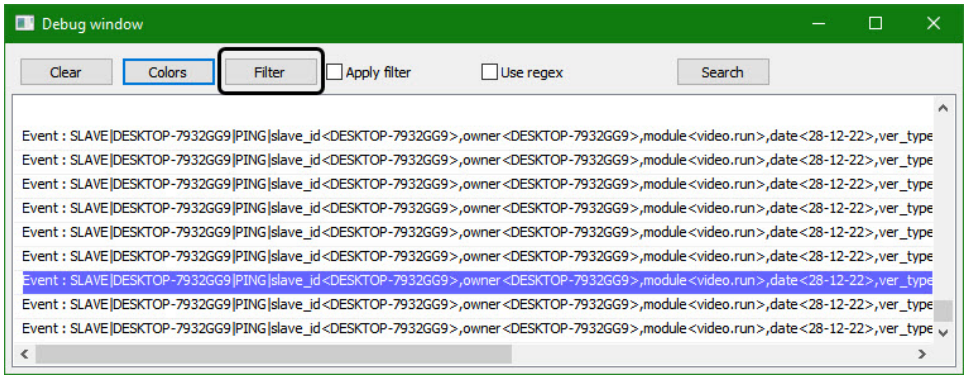
The messages are now highlighted.

Events and reactions filter

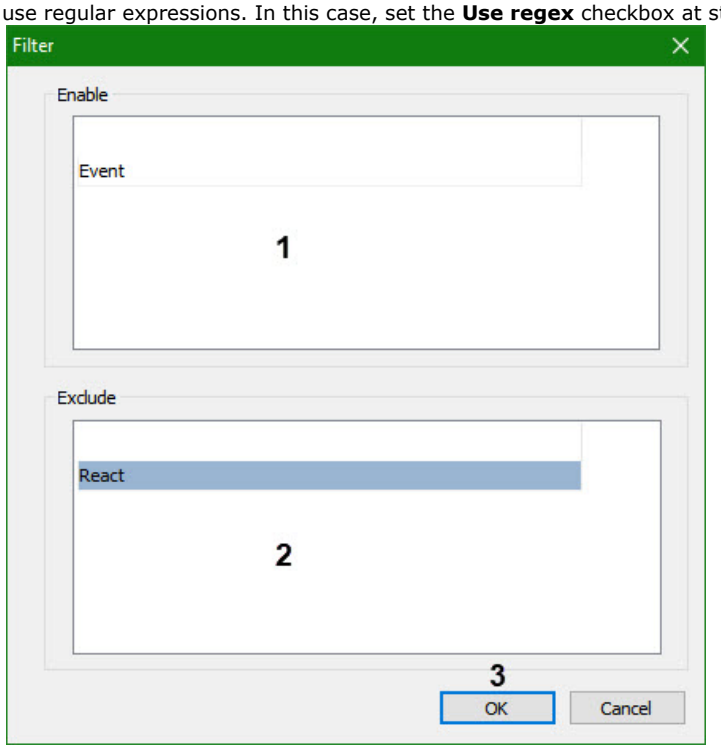
Events and reactions filter allows displaying only required messages in the **Debug window**.

To configure the events and reactions filter, do the following:

1. Click the **Filter** button.



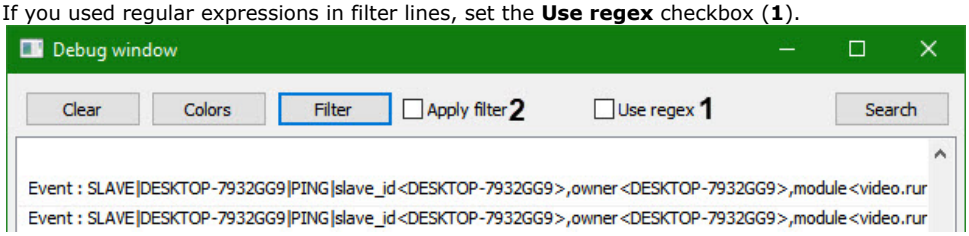
2. In the **Filter** window, specify the lines that must be in the message for it to be displayed in the **Debug window** (**1**). You can use regular expressions. In this case, set the **Use regex** checkbox at step 5.



3. Specify the lines that must be in the message for it not to be displayed in the **Debug window** (**2**). You can use regular expressions. In this case, set the **Use regex** checkbox at step 5.

> **Note**
>
> To add a line to the table, press the key on the keyboard.

4. Click the **OK** button (**3**).
5. If you used regular expressions in filter lines, set the **Use regex** checkbox (**1**).



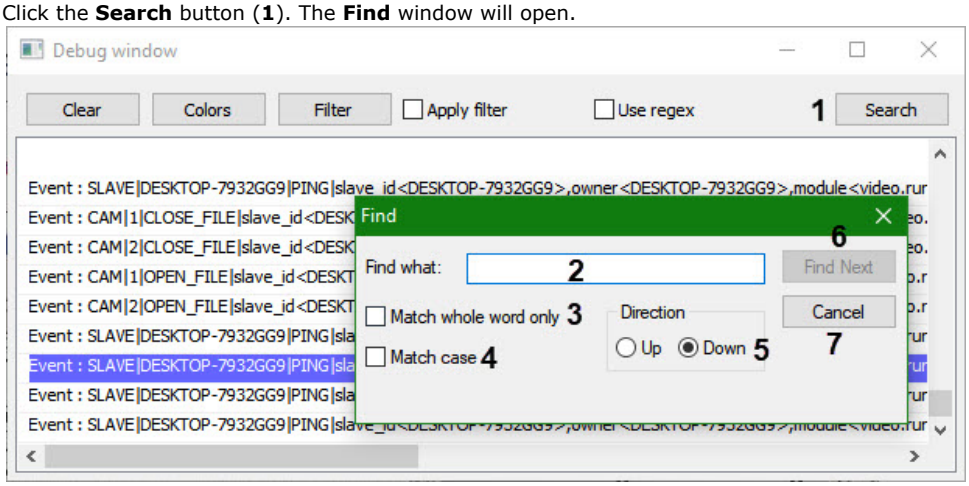
6. To apply the filter, set the **Apply filter** checkbox (**2**).

As a result, only messages that meet the filter conditions will be displayed in the **Debug window**.

Searching for events and reactions

To search for events and reactions, do the following:

1. Click the **Search** button (**1**). The **Find** window will open.



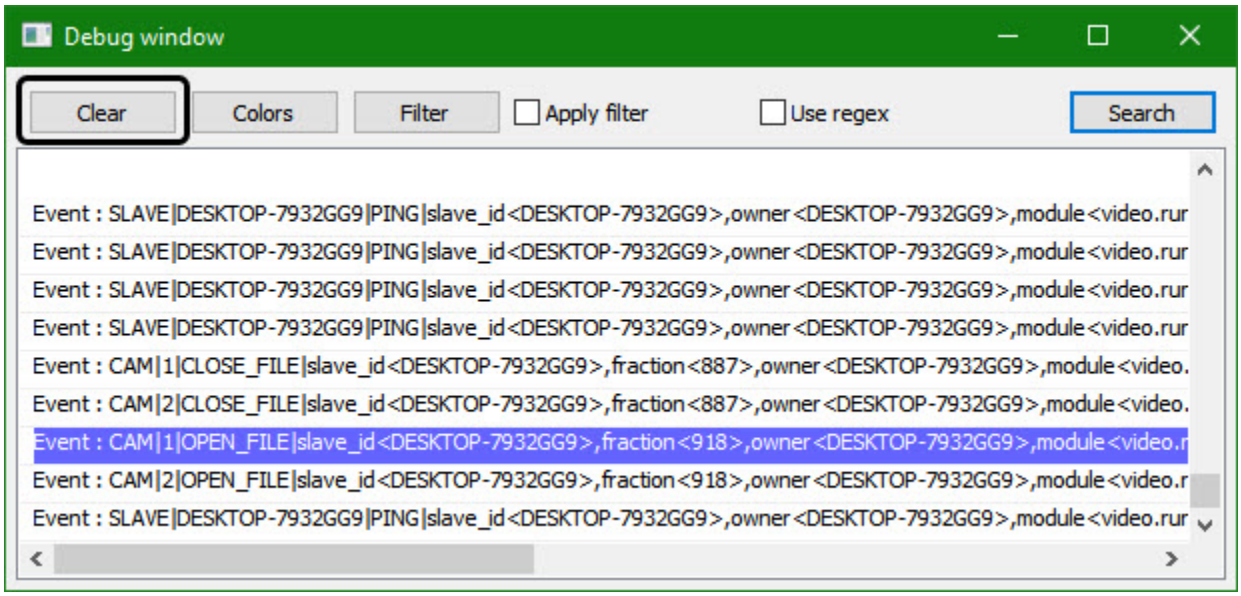
2. Specify the search criteria in the **Find what** field (**2**).
3. If you want to search for the entered string as an independent word, not present in other words as a part, but separated from them by at least one space, then set the **Match whole word only** checkbox (**3**).
4. If you want the search to be case sensitive, then set the **Match case** checkbox (**4**).
5. Set the **Direction** switch into the position corresponding to the search direction (**5**).
6. To view the next search result, click the **Find Next** button (**6**).

> **Note**
>
> To close the **Find what** window, click the **Cancel** button.

The search for events and reactions is now complete.

Clearing the Debug window

To delete all messages from the Debug window, click the **Clear** button.



Getting the list of system names of objects, reactions and events in Axxon PSIM

You can get the list of system names of objects, reactions and events used when programming with the help of the *ddi.exe* configuration setting utility. See the description of the main system objects reactions in Description of events and reactions of system objects.

The *ddi.exe* utility is started in one of the following ways:

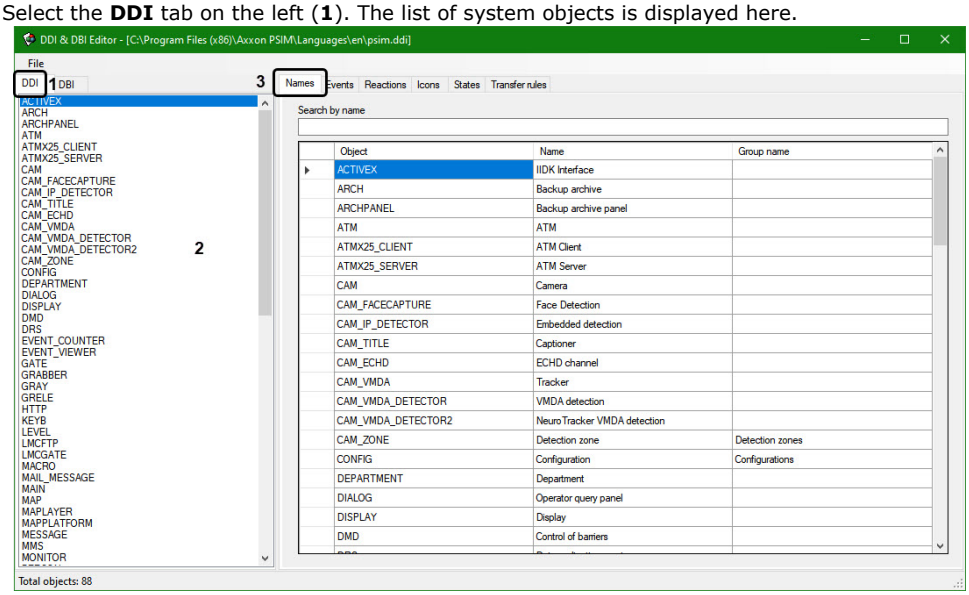
1. From the **Start** menu **Axxon PSIM System configuration**.

   > **Note**
   >
   > The *ddi.exe* utility is available in the **Start** menu with the following types of *Axxon PSIM* installation: Server, Administrator's workstation and Workstation for monitoring.

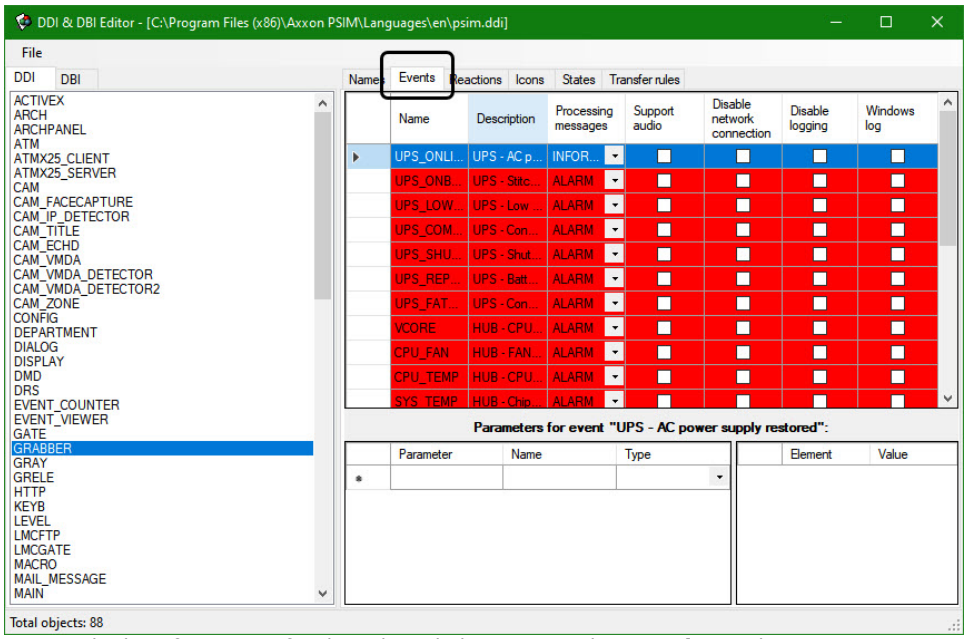
2. In the **Tools** folder of *Axxon PSIM* installation folder.

To view the list of system names of objects, events and reactions, do the following:

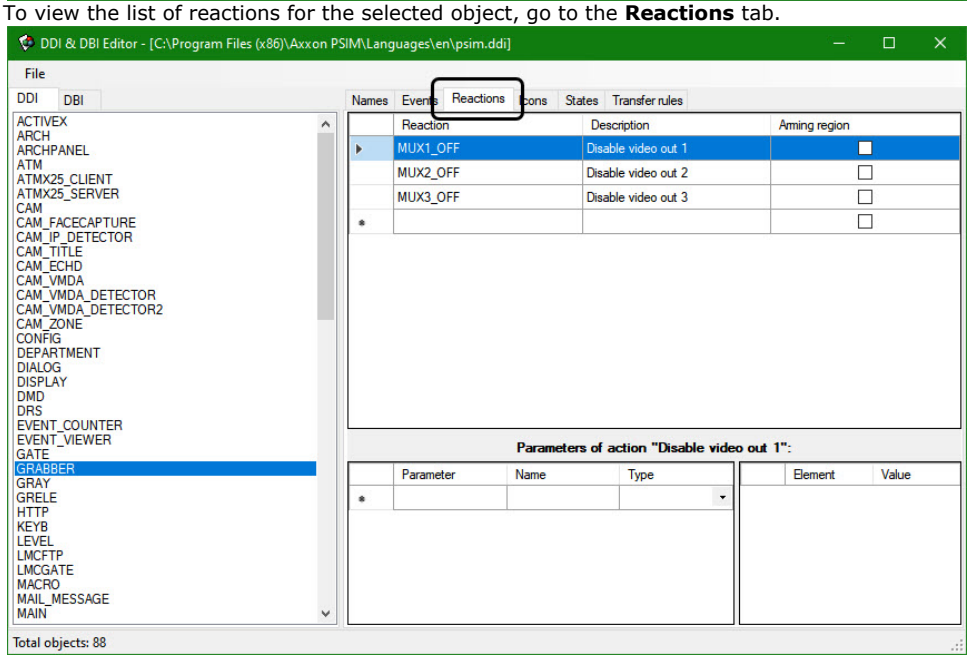
1. In the utility, open the *psim.ddi* file.
2. Select the **DDI** tab on the left (**1**). The list of system objects is displayed here.



3. In the **DDI** tab, select the object which events and reactions you want to view (**2**).
4. To view the name of the selected object, go to the **Names** tab (**3**).
5. To view the list of events for the selected object, go to the **Events** tab.



6.
7. To view the list of reactions for the selected object, go to the **Reactions** tab.



See the detailed information on working with the *ddi.exe* utility in Editing psim.dbi and psim.ext.dbi database templates using the ddi.exe utility.

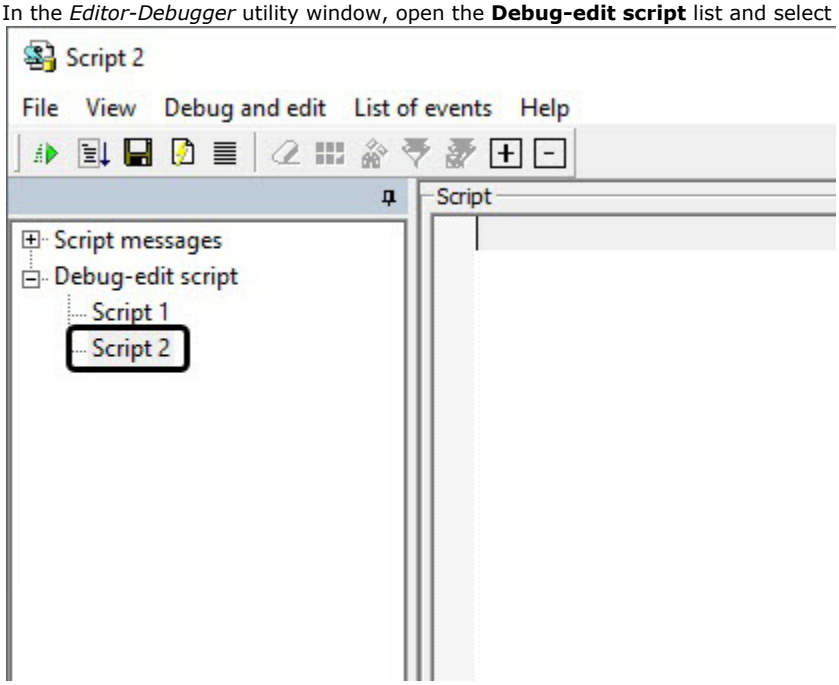
> **Note**
>
> If the Sensor is armed, then when the Sensor is closed/opened, the **Alarm** event occurs depending on the alarm mode setting (see the Creating and configuring the Sensor system object section of the Installing and configuring security system components guide). If the Sensor is disarmed, the **Closed**/**Opened** events occur correspondingly.

Creating your first script

As an example of using JScript in *Axxon PSIM*, try to create a script containing an error and then correct it. The script performs the following actions: when **Macro 1** starts, set the value 10 to the **"Hot record" time** parameter for cameras 1–4 and output the "Hello world" message to the debugging window of the *Editor-Debugger* utility.

To create and run this script, do the following:

1. In the **Hardware** tab of the **System Settings** dialog window, create four **Camera** objects with identification numbers 1, 2, 3 and 4, if they have not been created before.
2. In the **Programming** tab, create a **Macro** object with identification number 1. Don't fill in the **Events** table for the correct execution of the following actions and successful run of the script.
3. In the **Programming** tab, create a **Script** system object. Give the object the identification number 1 and the name "Script 1".
4. On the settings panel of the **Script 1** object, select **Always** from the **Time schedule** drop-down list.
5. Click the **Editor-Debugger** button at the bottom of the settings panel of the **Script 1** system object. The *Editor-Debugger* utility window will open.
6. In the *Editor-Debugger* utility window, open the **Debug-edit script** list and select the **Script 2** object.



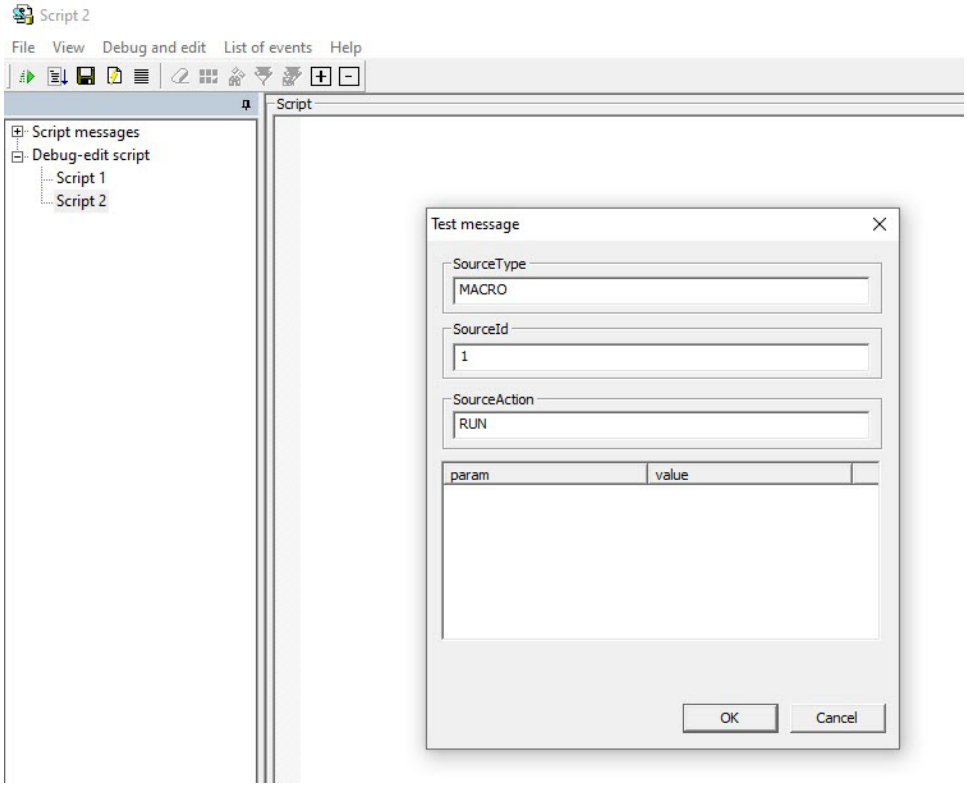
7. In the **Script** field, enter the following:

```
 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");
}
```

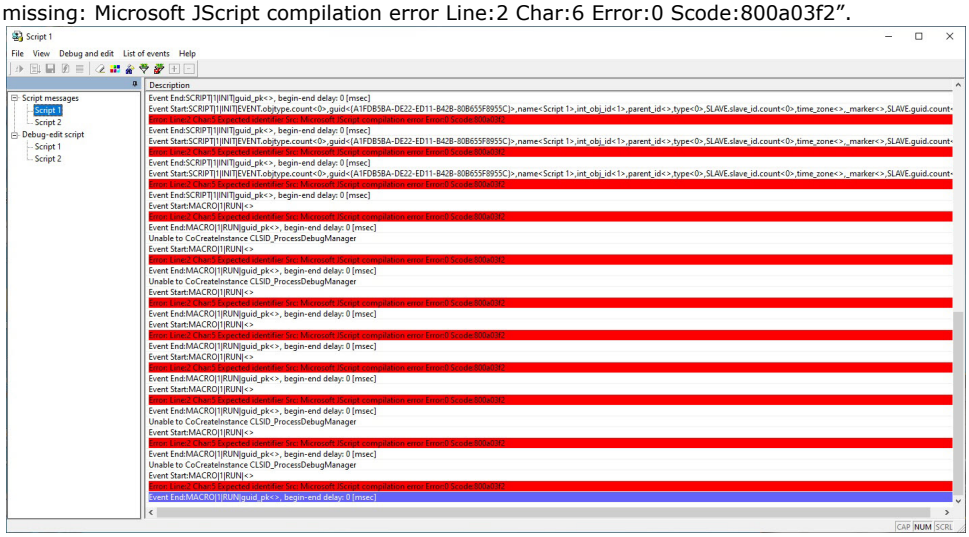
> **Attention!**
>
> The script contains an error. See below the recommendations on how to fix it.

8. In the **File** menu, select **Save to database** to save the script.

9. Create a test event to run the script in debug mode—MACRO|1|RUN|. To achieve this, in the **Debug and edit** menu, select **Edit test event**. The **Test message** window will open. Fill in the fields in the **Test message** window as shown in figure.



10. To run the script with the test event, select **Test run** in the **Debug and edit** menu.
11. Open the **Script Messages** list and select **Script 1**. The debugging window will open on the right side.
12. In the debugging window, find the "Process Event:MACRO|1|RUN|" line and the following error message: "Src identifier missing: Microsoft JScript compilation error Line:2 Char:6 Error:0 Scode:800a03f2".



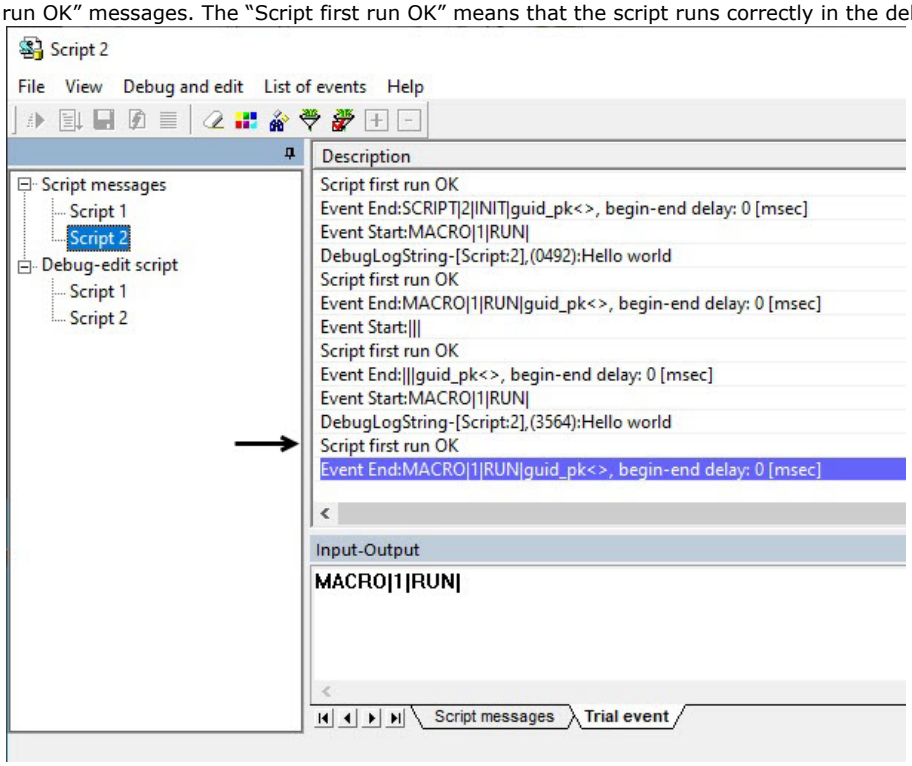
The error message says that there is no identifier in the second line of variable declaration operator (var). This means that no variable has been declared. This is a critical error in JScript, thus the script has not been executed.

13. Correct the text of the script (see the "var i;" line).

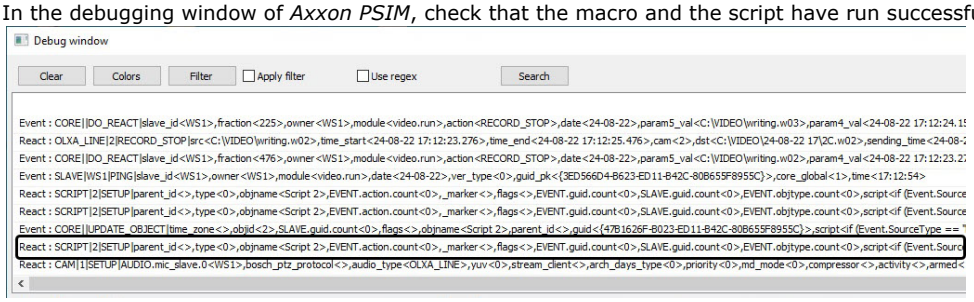
```
 if (Event.SourceType == "MACRO" && Event.SourceId &&
Event.Action == "RUN")
{
 var i;
 for(i=1; i<=4; i=i+1)
 {
 SetObjectParam("CAM",i,"hot_rec_time","10");
```

```
 }
DebugLogString ("Hello world");
}
```

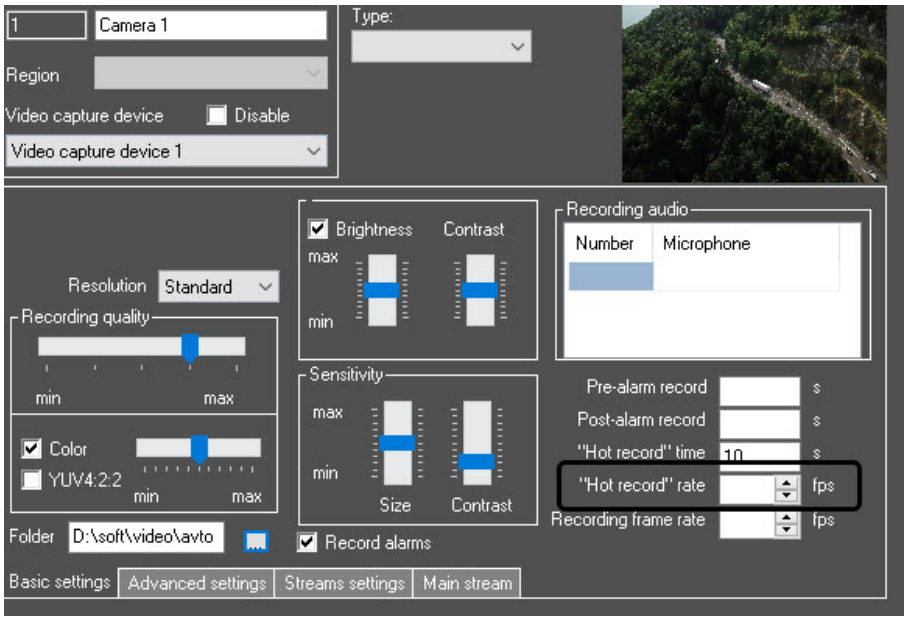
14. In the **File** menu, select **Save to database** to save the script.
15. Repeat steps 10 and 11.
16. In the debugging window, find the "Process Event:MACRO|1|RUN|" line and the "DebugLogString:Hello world" and "Script first run OK" messages. The "Script first run OK" means that the script runs correctly in the debug mode.



17. Close the *Editor-Debugger* utility.
18. The text of the created script will be displayed in the field of the **Script 1** system object. Click the **Apply** button on the settings panel of the **Script 1** system object to activate the script.
19. Select **Macro 1** in the **Run** menu of the main control panel.
20. In the debugging window of *Axxon PSIM*, check that the macro and the script have run successfully.



21. Check the accuracy of the script result. The **"Hot record" time** field in the **Camera 1** to **Camera 4** objects settings panels must have the 10 value.



> **Note**
>
> The **"Hot record" time** field in the **Camera** settings panel is blank by default.

Script creation and debugging is now complete.

Working with script

Creating a script

On the page:

To create and run scripts in JScript, create the **Script** system object. Then, in the *Editor-Debugger* utility, enter the script, check it and debug it.

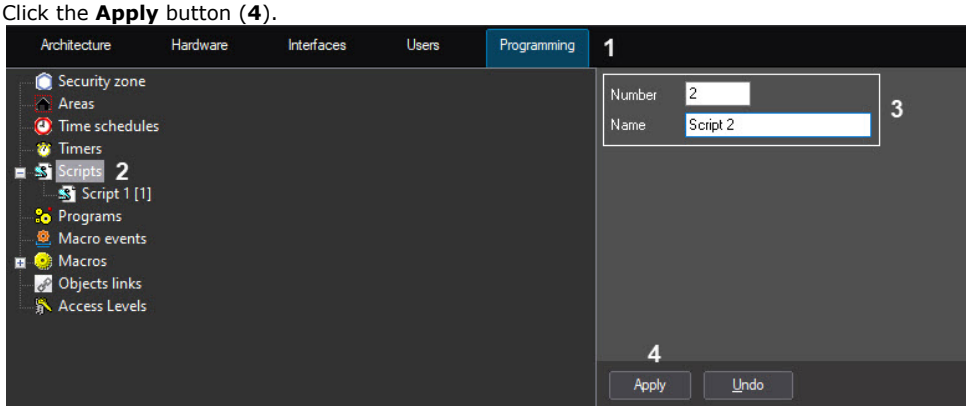
You can create JScript scripts in *Axxon PSIM* using the built-in *Editor-Debugger* utility.

To start the *Editor-Debugger* utility, click the **Editor-Debugger** button on the settings panel of the **Script** system object.

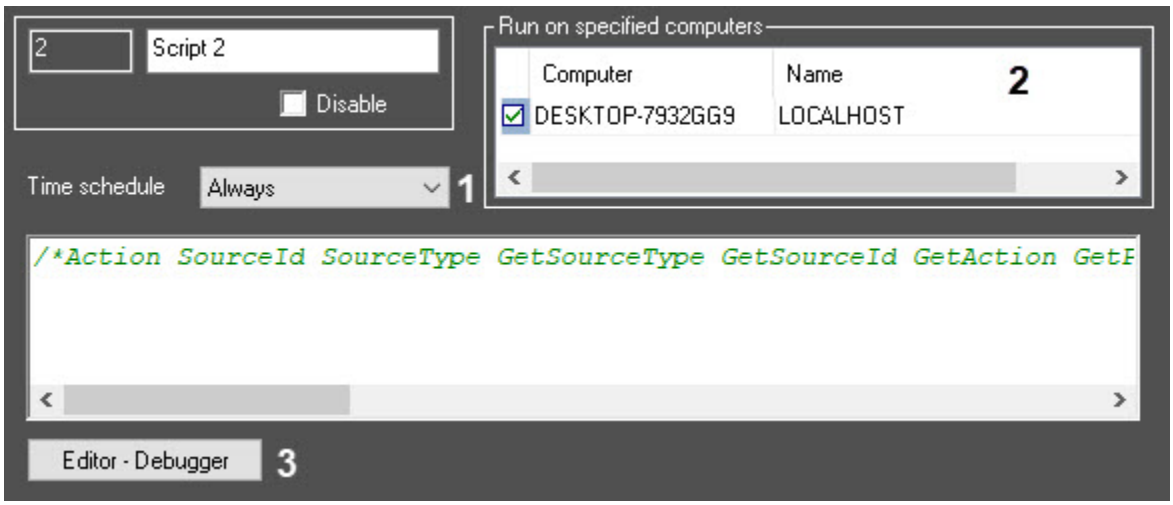
Creating the Script object

To create the **Script** object in *Axxon PSIM*, do the following:

1. In the **Programming** tab (**1**) of the **System settings** dialog window, create the **Script** object (**2**).
2. Enter the identification number and the name for the **Script** object (**3**).
3. Click the **Apply** button (**4**).



The settings panel of the **Script** object will open.



To configure the **Script** object, set the values of the following parameters:

1. In the **Time schedule** field (**1**), specify the time schedule of the script execution: **Always**, **Never** or one of the schedules created earlier (1, see Creating and configuring Time schedules). The default value is **Never**.
2. In the **Computers** field (**2**), select the computers (kernels) on which the script must run.

> **Note**
>
> By default, the script will run on all computers (kernels). The list displays only the computers registered in the **Hardware** tab of the **System settings** dialog window.

3. Click the **Apply** button.

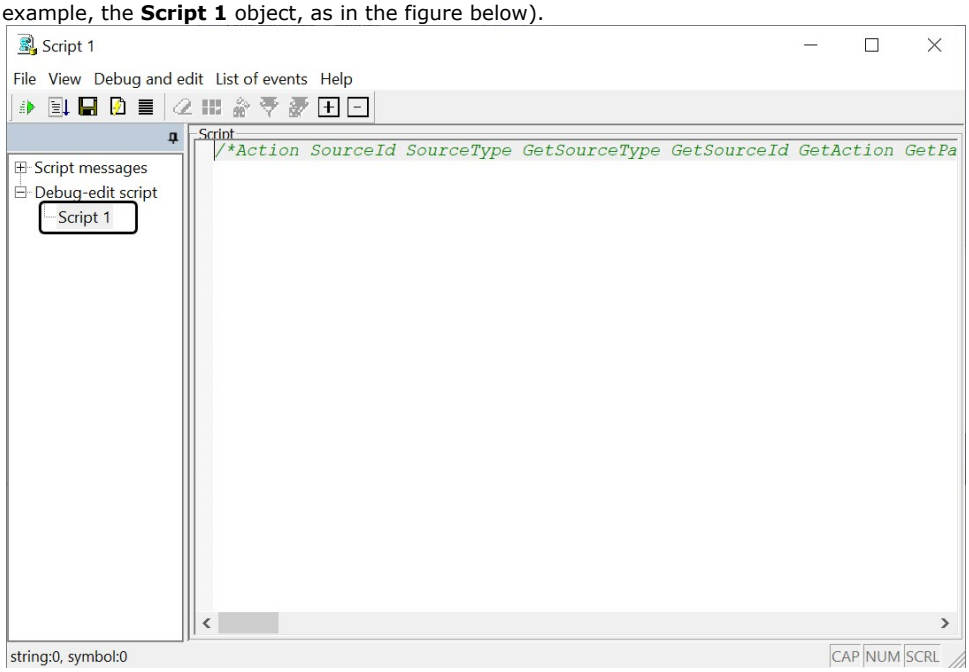
Creating and editing a script

To create a script in JScript in *Axxon PSIM*, do the following:

1. Click the **Editor-Debugger** button (**3**) at the bottom of the **Script** system object panel to open the *Editor-Debugger* utility.

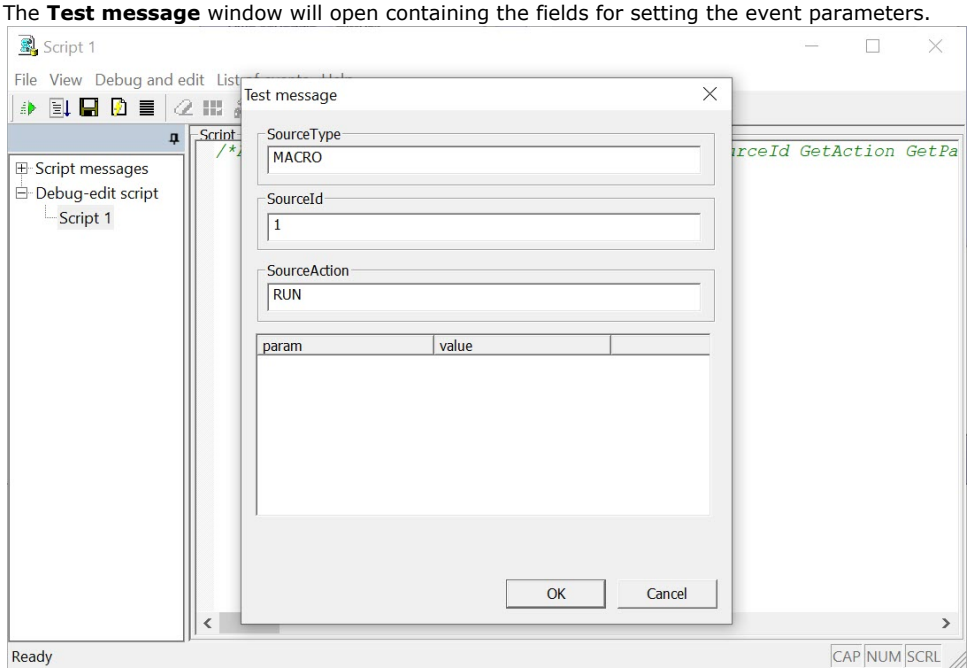
> **Note**
>
> Use the *Editor-Debugger* utility to create, edit and save scripts in JScript. The settings panel of the **Script** system object displays the text of the saved script that can also be edited.

2. In the *Editor-Debugger* utility window, open the **Debug-edit script** list and select the **Script** object you want to edit (for example, the **Script 1** object, as in the figure below).



3. In the **Script** field, enter the text of the script in JScript programming language (see Examples of scripts in the JScript language).

4. Run the script using a test event. To create a test event, select **Debug and edit Edit test event**. The **Test message** window will open containing the fields for setting the event parameters.



To run the script using the test event, select **Debug and Edit Test run**.

> **Note**
>
> You can also run the script using the test event using the **Ctrl+T** key combination.

Features when working with a script

When working with a script in the *Editor-Debugger* utility, you can undo the last action or return the last action. To undo the last action, press the **Ctrl+Z** key combination. To return the last action, press the **Ctrl+Y** key combination.

> **Note**
>
> For your convenience, when editing the script, the cursor position is saved when saving the script or when switching between the scripts in the *Editor-Debugger* utility window during a session, it means, until *Axxon PSIM* is restarted.

> **Note**
>
> When you go to the **Script messages** list when editing a script, and then go back to the corresponding script in the **Debug-edit script** list, you cannot undo or return the last action.

Debugging a script

You can check if the script syntax is correct using the interpreter which is built-in into the *Editor-Debugger* utility. The result of the check with the information about the content and location of the error is displayed in the **Debugger window** corresponding to the script in the **Script messages** list. If there are errors, you need to edit the script syntax and check it again.

> **Note**
>
> See the detailed information about using test events for script debugging in Script debugging.

After debugging the script using the *Editor-Debugger* utility, run it with a real system event. Check the result of the script execution. If the result is incorrect, make the necessary changes and run the script again.

Script creation is considered complete if it runs correctly.

Saving a script

The *Editor-Debugger* utility provides two options for saving scripts: in the **Script** system object, or in a text file on the hard drive.

To save the script in the **Script** system object, in the **File** menu, select **Save in the database**. When you select this menu item, the script will be updated on every Server that you selected on settings panel of the **Script** object (see The Script system object).

> **Note**
>
> You can also save the script in the database using the **Ctrl+S** key combination.

> **Note**
>
> The script is automatically saved in the corresponding **Script** system object upon closing the *Editor-Debugger* utility.

To save the script in the file, in the **File** menu, select **Save on disk**. To open a script saved in a file in the *Editor-Debugger* utility, in the **File** menu, select **Open from disk**.

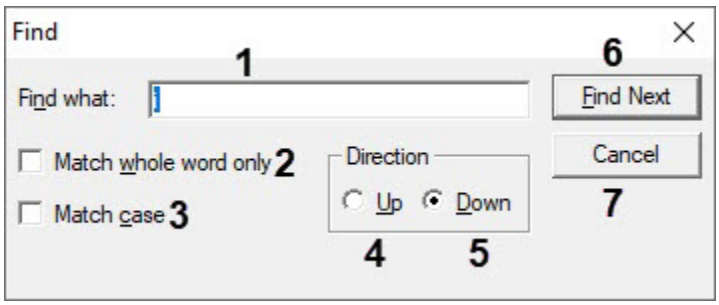
Deleting a script

To delete a script created in *Axxon PSIM*, delete the corresponding **Script** system object in the **Programming** tab.

Searching text in script

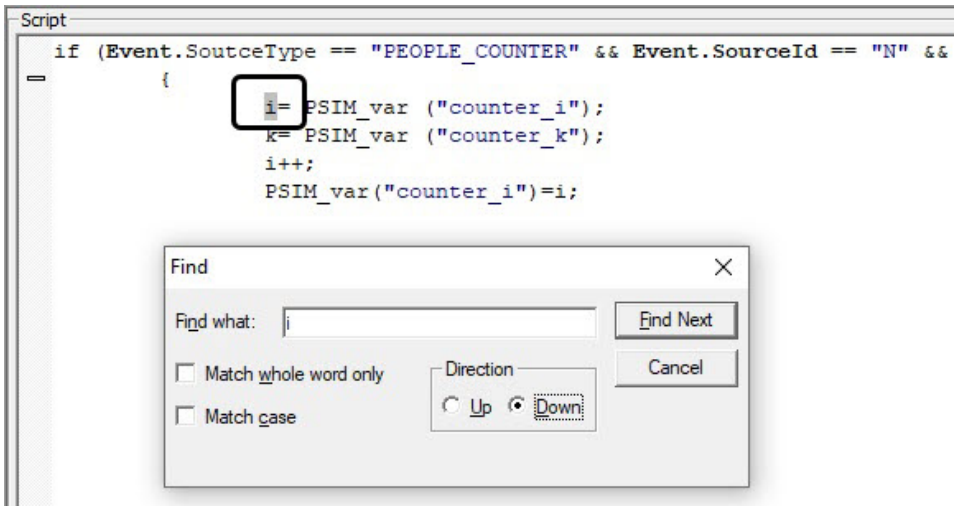
The *Editor-Debugger* utility allows searching text in script using a dialog box.

Press **Ctrl+F** to open the search dialog box. The **Find** dialog box will open.



1. In the **Find what** field (**1**), enter the text that you want to find.
2. Set the **Match whole word only** checkbox (**2**) if you want to search for the whole entered text.
3. Set the **Match case** checkbox (**3**) if you want the search to be case-sensitive.
4. Select the direction of the search through the script relative to the current cursor position: **Up** (**4**) or **Down** (**5**).
5. Click the **Find Next** button (**6**) to start the search and go to the next match.
6. Click the **Cancel** button (**7**) to cancel the search.

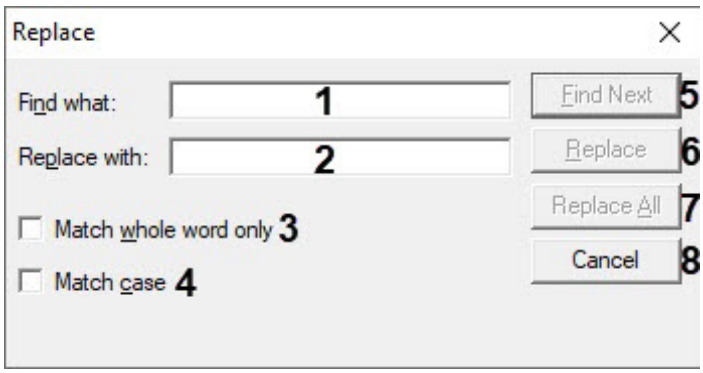
The found match is highlighted in the *Editor-Debugger* utility window.



Replacing text in script

The *Editor-Debugger* utility allows replacing text in script using a dialog box.

Press **Ctrl+H** to open the **Replace** dialog box.



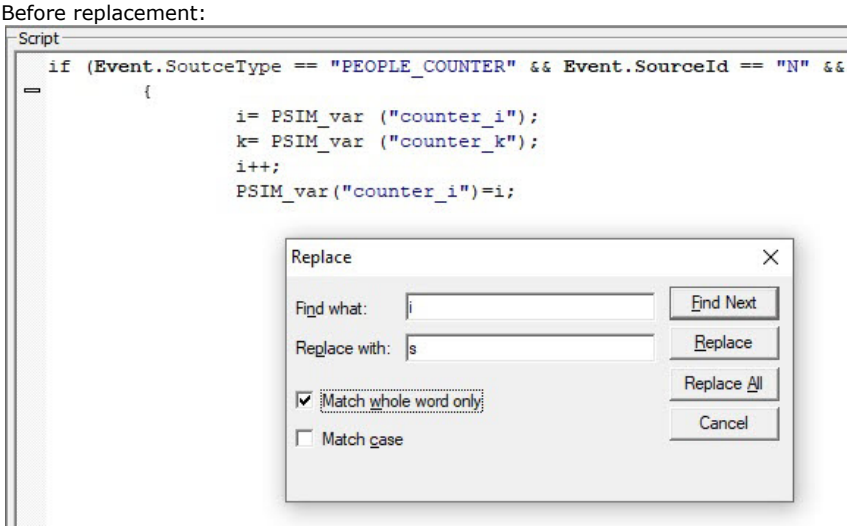
1. In the **Find what** field (**1**), enter the text that you want to find in the script.
2. In the **Replace with** field (**2**), enter the text with which you want to replace the found text in the script.
3. Set the **Match whole word only** checkbox (**3**) if you want to search for the whole entered text.
4. Set the **Match case** checkbox (**4**) if you want the search to be case-sensitive.
5. Click the **Find Next** button (**5**) to start the search and go to the next match.

> **Note**
>
> The search runs down from the current cursor position.

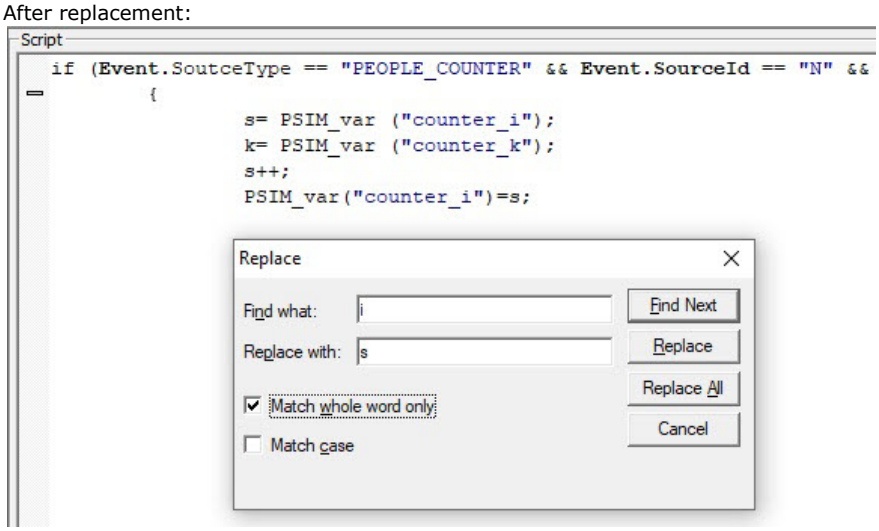
6. Click the **Replace** button (**6**) to replace the current found match.
7. Click the **Replace All** button (**7**) to replace all matches automatically.
8. Click the **Cancel** button (**8**) to close the replace dialog box.

Example of replacing the **i** variable with **s**:

1. Before replacement:



2. After replacement:



Script debugging

Script debugging features

The *Editor-Debugger* utility allows debugging scripts using the built-in tools for checking script syntax, script interpreting and script execution with test events generated by the utility. The messages about the debugging results are displayed in the corresponding debugging windows.

The *Editor-Debugger* utility provides the following debugging functionality:

1. A separate debugging window is assigned to each **Script** object, in which the test and system events, error messages, success messages and user information messages are displayed. The messages in the debugging windows can be filtered.
2. Special **Information window** debugging windows are available that display the messages related to the script being debugged.
3. Test events generated by the *Editor-Debugger* utility, which are not registered in the system, are used to check script accuracy.
4. Third-party debugging programs can be used for step-by-step script execution, viewing script variables during execution, and so on.

Creating and using test events

On the page:

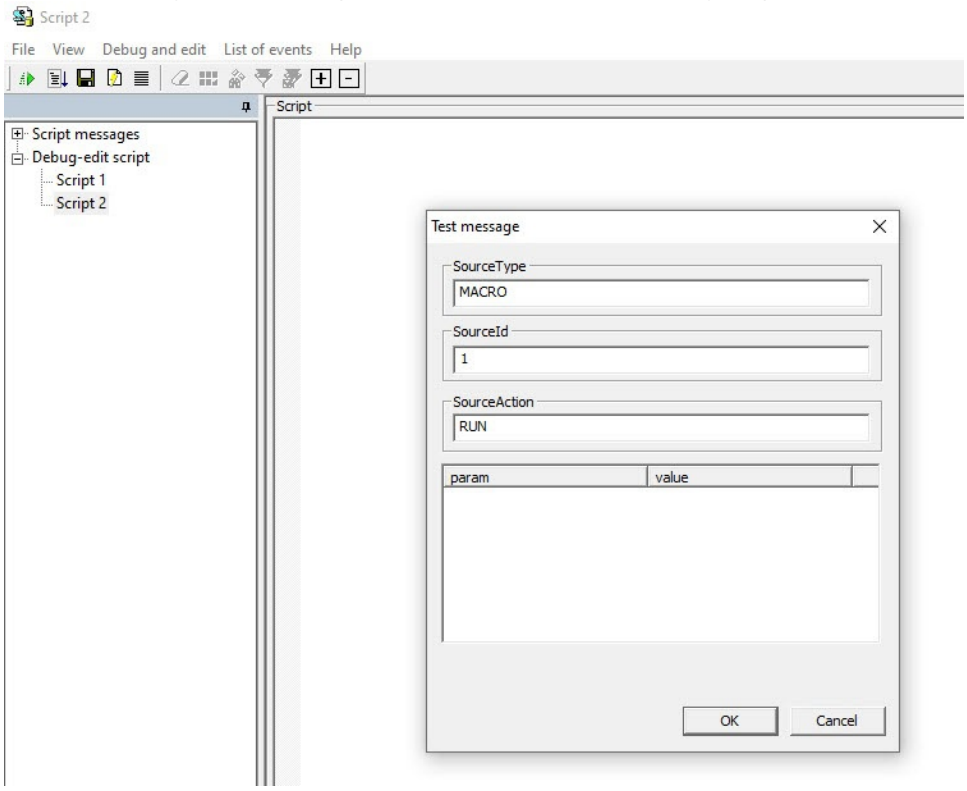
Creating test events

The *Editor-Debugger* utility can use test events specified by the user and generated by the utility to debug scripts. Test events are not registered by the video surveillance system, it means they aren't displayed in the Event Viewer and aren't saved to the database.

No more than one test event can be created for each script.

To create a test event, do the following:

1. In the **Debug and edit** menu, select **Edit test event**, or click the  button in the toolbar.
2. The **Test message** window will open. This window is used for entering the parameters of a test event.



3. Enter the following information in the fields of the **Test message** window:
   a. **SourceType**—system object type;
   b. **SourceId**—system object identification number;
   c. **SourceAction**—event generated by the specified system object;
   d. **param**—additional event parameters;
   e. **value**—values of the additional parameters.

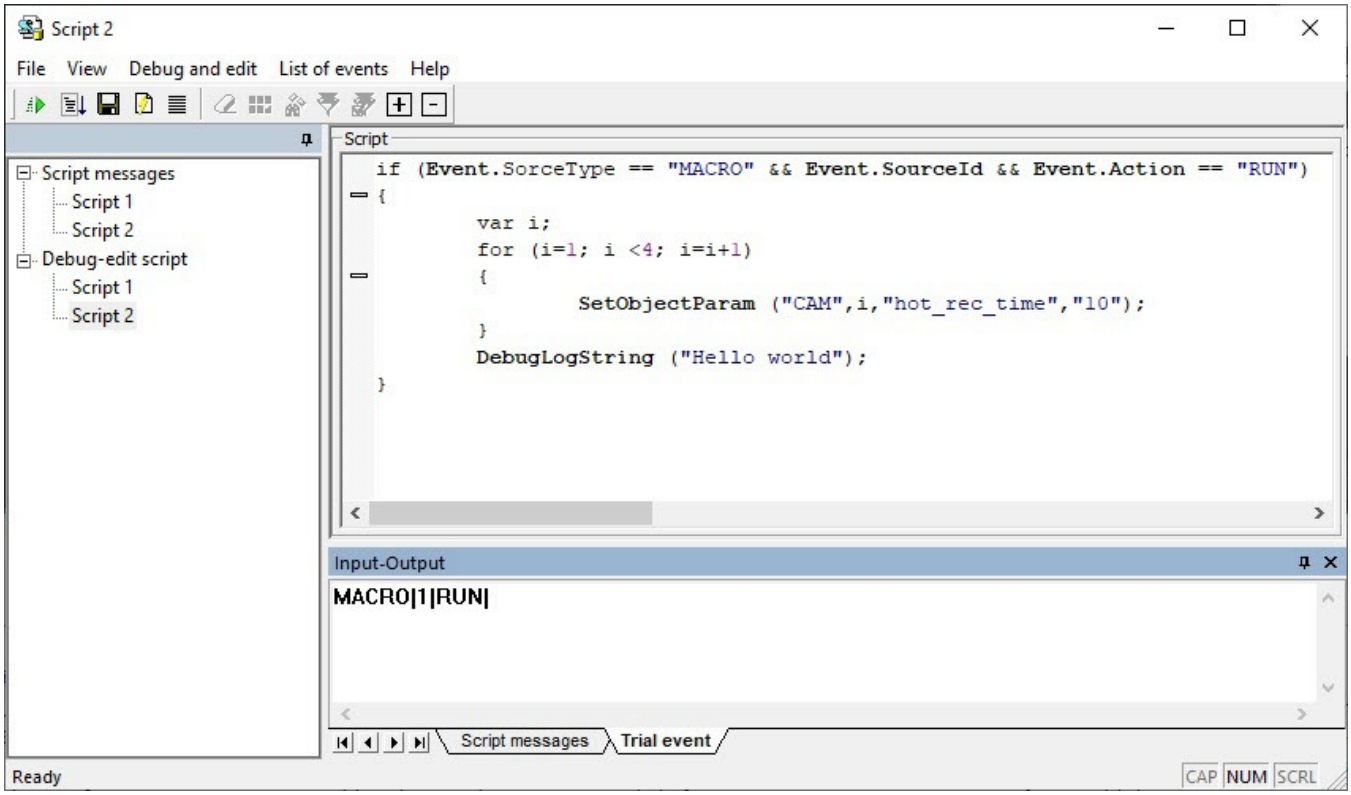
> **Note**
>
> The main system objects, their events and parameters are described in Description of events and reactions of system objects. You can also get them using the ddi.exe utility (see Getting the list of system names of objects, reactions and events in Axxon PSIM).

4. When you filled in the fields of the **Test message** window, click the **OK** button.

The test event is now created.

The created test event will be displayed in the **Event** field in a special string format.

The figure below shows the test event **Arm Camera 111**.



Running a script with a test event

To run the script with a test event, do one of the following:

1. Click the **Test run** button in the toolbar.
2. In the **Debug and edit** menu, select **Test run**.
3. In the **Debug and edit** menu, select **Test run in third-party debugger**.

When you select the **Test run in third-party debugger** option, the third-party debugger starts to run the test (see Using third-party debugger programs).

The results of the verification and execution of the script are displayed in the corresponding debugging window of the *Editor-Debugger* utility.

Working with the debugging windows of the Editor–Debugger utility

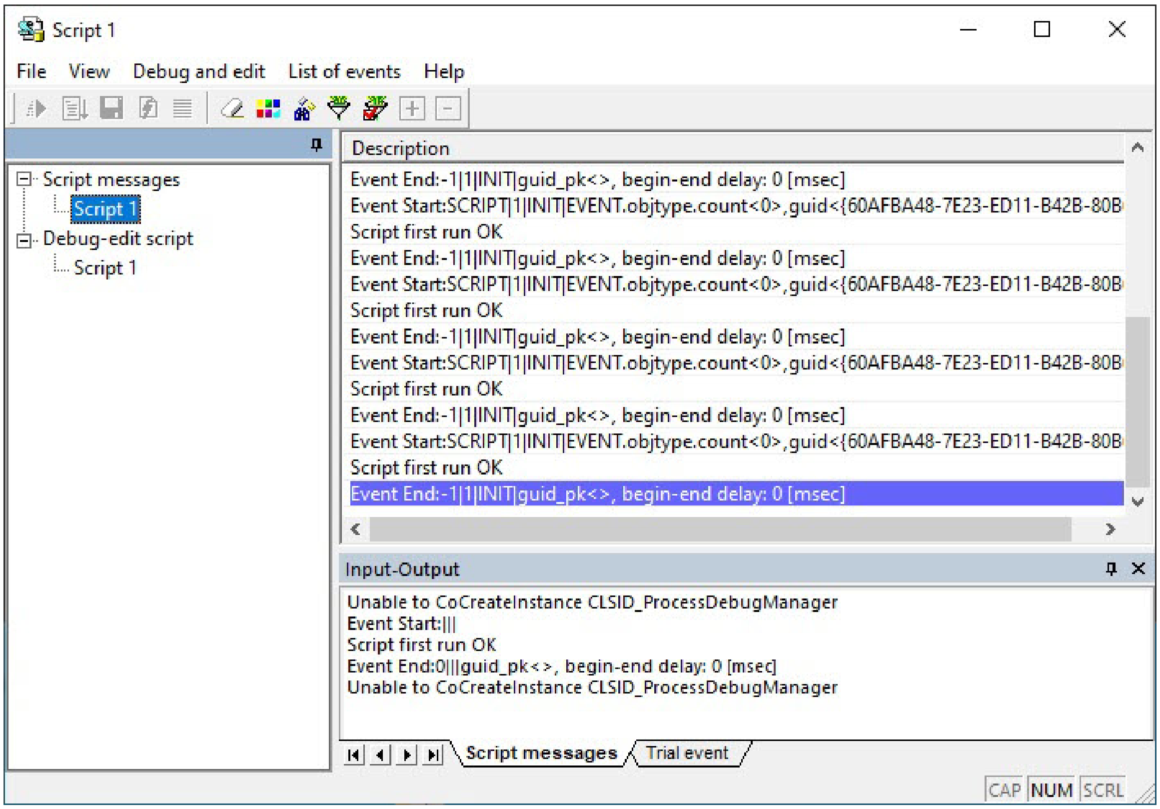
Viewing the script messages

Debugging windows display messages about logging system and test events, errors and successful script execution, as well as user information messages.

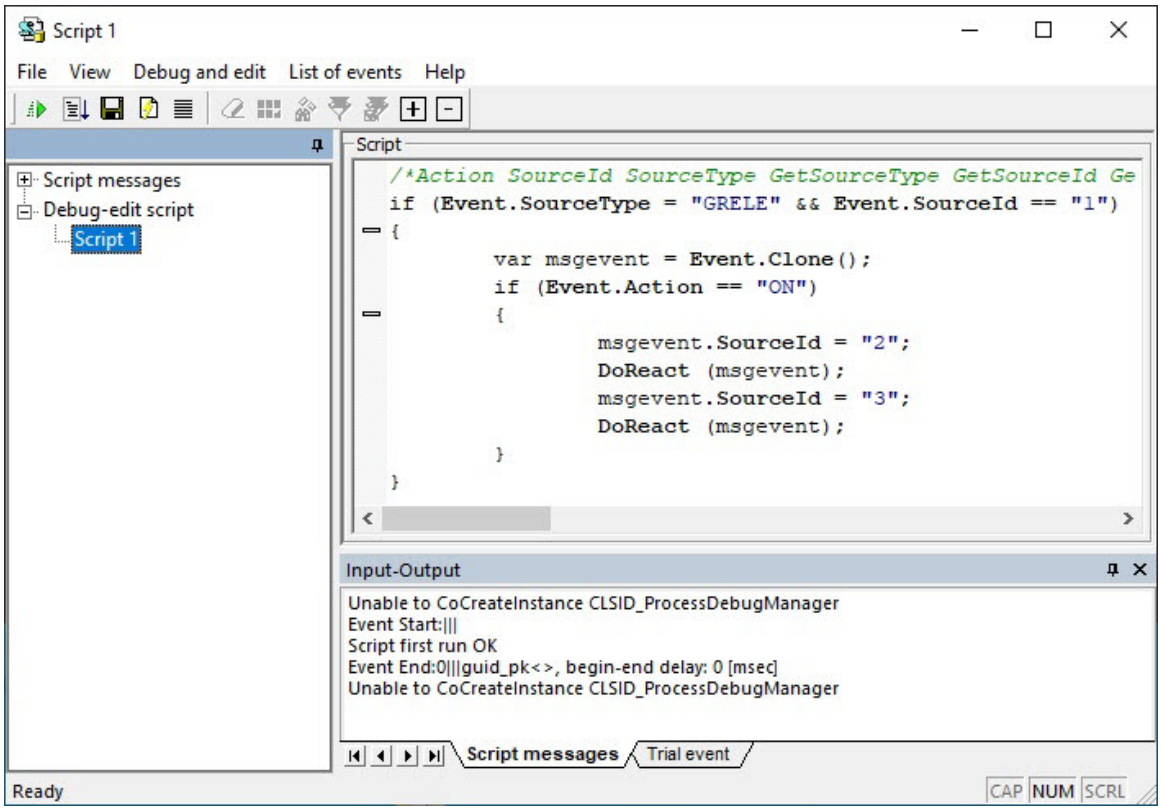
A separate debugging window is assigned to each script in the *Editor-Debugger* utility.

There are two types of debugging windows: **All script messages** and **Last run script messages**.

The debugging windows of the **All script messages** type are displayed in the **Script messages** list. The names of the debugging windows match the names of the corresponding **Script** objects. These windows display all system messages related to the corresponding script. The example of the debugging window of the **All script messages** type:



In addition, the information on the last script run is displayed on the **Script messages** tab at the bottom of the *Editor-Debugger* utility window. If this panel is not displayed, select **Debug and edit Summary Information**, or click the  button on the toolbar.



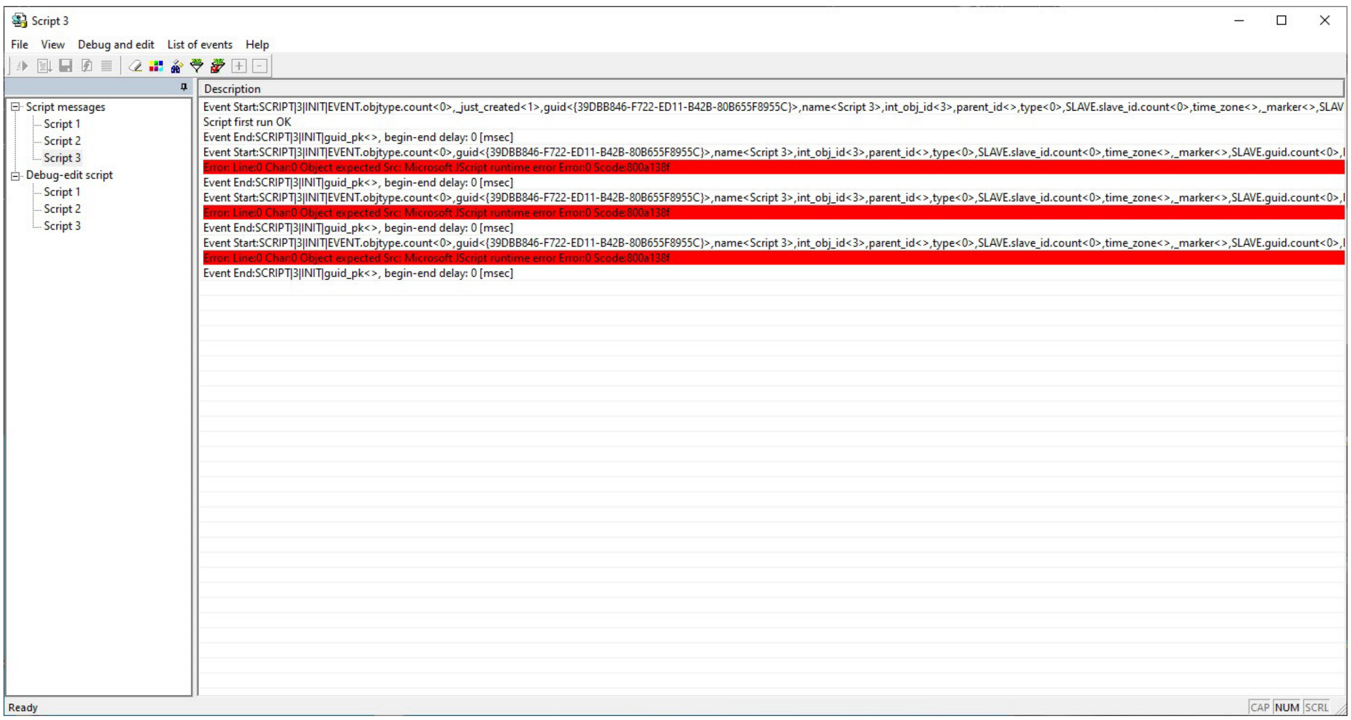
Both types of debugging windows are used in the same way.

Displaying messages about starting, verifying, changing and executing scripts in the debugging windows

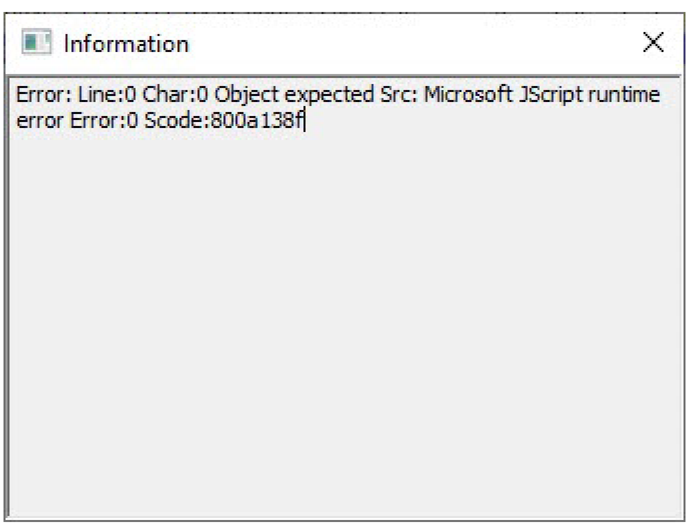
The messages in the debugging window track the stages of starting, verification and execution of scripts.

When the event that triggers the script occurs, the following message is displayed in the debugging window: “Process Event: <script triggering event>”. For example, if the script starts on Macro 1, the line reads “Process Event:MACRO|1|RUN|”. At the moment of changing the script in the *Editor-Debugger* utility or in *Axxon PSIM,* the debugging window displays the message «Process Event: SCRIPT|script’s number|SETUP|» (for example, when changing the script with number 1, the debugging window displays «Process Event: SCRIPT|1|SETUP|»).

Script syntax is checked before execution. If there are syntax errors, related error messages will be displayed in the debugging window. The figure below shows an example of a syntax error message.



Right-click the message to view its complete text. The **Information** window will open containing the full text of the error message.



The message contains the following information:

1. error description;
2. error type (for example, **Src: Microsoft Jscript compilation error**);
3. error location in the script text (line number and character number in the "Char" line);
4. error code (**Scode**).

If there are no syntax errors in a script, the following message will be displayed in the debugging window: **Script first run OK**. Then, the script will run.

Script runtime errors are also displayed in the debugging window.

In case of successful execution of the script, the following message will be displayed: “End ProcessScript, begin-end delay: <script execution time>; for example, “End ProcessScript, begin-end delay: 13 [msec]”.

Using third-party debugger programs

Axxon PSIM officially supports *Microsoft Visual Studio 2005* debugger.

Axxon PSIM allows using third-party debuggers for processing JScript scripts. These programs may have the functionality that is not included in the *Editor-Debugger* utility, for example, step-by-step script execution, watching script variables during script execution, and so on.

> **Note**
>
> You must use third-party debuggers with caution, as they don't provide full compatibility with *Axxon PSIM*. The use of third-party debuggers may lead to failure of *Axxon PSIM.*

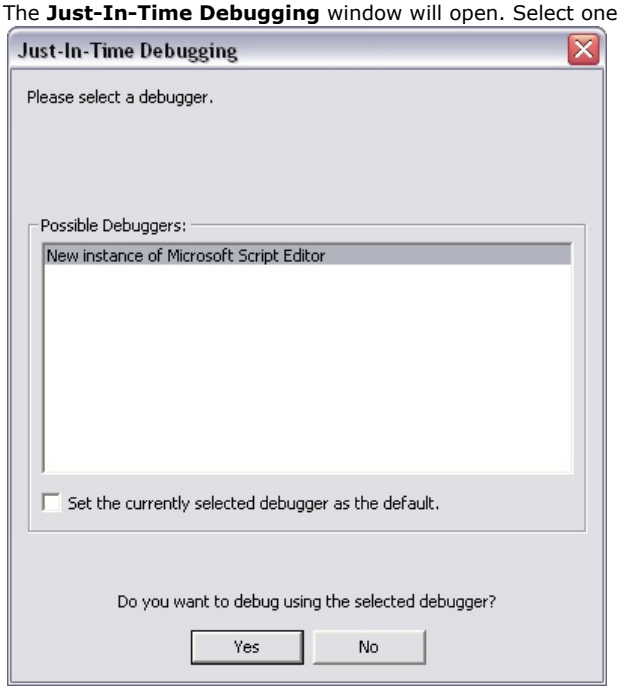
We strongly recommend introducing the breakpoint in the script when using third-party debuggers. To insert the breakpoint, add the debugger; command to the script. The script execution will pause at the place specified by the debugger; command, and the debugger will start automatically.

> **Note**
>
> In programming, a breakpoint is a deliberate interruption of program execution at which a debugger call is made.

When you use a third-party debugger, scripts can be started with test events only.

To start the script using a third-party debugger, do the following:

1. Create a script and add the debugger; command to it.
2. Create a test event to run the script.
3. In the **Debug and edit** menu, select **Test run in third-party debugger**.
4. The **Just-In-Time Debugging** window will open. Select one of the debuggers installed on the computer.



5. Click **Yes** to confirm the selection.
6. In case of a successful syntax check (no errors found before the “debugger;” line), the third-party debugger will start. The script will pause at the breakpoint.

Example. A script with the breakpoint after Macro 1 starts.

```
 if (Event.SourceType== "MACRO" && Event.SourceId=="1" && Event.Action == "RUN"); //start Macro 1
{
debugger; // breakpoint
DebugLogString ("Hello world");
}
```

Examples of scripts in the JScript language

To illustrate the available fields of application of scripts in JScript, see the following examples, which can be used to create additional functions in the system on the basis of the **Script** object.

Examples of scripts with Video surveillance monitor and Cameras

MONITOR Monitor

CAM Camera

Example 1. Visualisation of operating the Queue length detection in the Video surveillance monitor

For the script to work correctly, you must first create and configure the **Queue length detection** object (part of the Detector Pack package), **Camera** and **Captioner** objects (below, instead of the N, M, L characters, set the corresponding numbers of the Queue length detection, Camera and Captioner objects) in *Axxon PSIM.*

```
//Event reading by the queue length
if (Event.SourceType == "OCCUPANCY_COUNTER" && Event.SourceId == "N" && Event.Action == "OCCUPANCY")  //N -
Number of Queue length detection
{
  var n=Event.GetParam("occupancy");
//Displaying the queue length by the Captioner in the Monitor
  DoReactStr("CAM","M","CLEAR_SUBTITLES","title_id<L>"); //M - Number of Camera  L - Number of Captioner
  DoReactStr("CAM","M","ADD_SUBTITLES","command<Queue length: "+n+" person(s).\r>,page<BEGIN>,title_id<L>");
//M, L - the same
}
```

As a result, when the corresponding camera is displayed in the Monitor, a text message about the current queue length will be superimposed on the video image.

You can configure the font, color and position of text on the settings panel of the **Captioner** object.

Note

When you use the page<BEGIN> and page<END> parameters, the corresponding fields of the captions database are filled in. This enables data search using the **Captions search** interface object.

Example 2. Visualisation of operating the People counter detection in the Video surveillance monitor

For the script to work correctly, you must first create and configure the **People counter detection** object (part of the Detector Pack package), **Camera**, **Captioner** and **Macro** objects (below, instead of the N, M, L, P characters, set the corresponding numbers of the People counter detection, Camera, Captioner and Macro objects) in *Axxon PSIM*.

```
//Event reading and counting of entered people
if (Event.SourceType == "PEOPLE_COUNTER" && Event.SourceId == "N" && Event.Action == "IN") //N - Number of
People counter detection
  {
    i = Itv_var("counter_i");
    k = Itv_var("counter_k");
    i++;
    Itv_var("counter_i")=i;
//Displaying the number of people by the Captioner in the Monitor

    DoReactStr("CAM","M","CLEAR_SUBTITLES","title_id<L>"); //M - Number of Camera  L - Number of Captioner
    DoReactStr("CAM","M","ADD_SUBTITLES","command<Number of people (entering/exiting): "+i+" / "+k+"\r>,
page<BEGIN>,title_id<L>"); //M, L - the same

  }
//Event reading and counting of exitng people
if (Event.SourceType == "PEOPLE_COUNTER" && Event.SourceId == "N" && Event.Action == "OUT") //N - Number of
People Counter detection
  {
    i = Itv_var("counter_i");
    k = Itv_var("counter_k");
    k++;
    Itv_var("counter_k")=k;
//Displaying the number of people by the Captioner in the Monitor

   DoReactStr("CAM","M","CLEAR_SUBTITLES","title_id<L>"); //M - Number of Camera  L - Number of Captioner
   DoReactStr("CAM","M","ADD_SUBTITLES","command<Number of people (entering/exiting): "+i+" / "+k+"\r>,
page<BEGIN>,title_id<L>"); //M, L - the same
  }
//Null the counter on Macro (the Macro must be created in Axxon PSIM beforehand)
if (Event.SourceType == "MACRO" && Event.SourceId == "P" && Event.Action == "RUN") //P - Number of Macro
  {
    Itv_var("counter_i")=0;
    Itv_var("counter_k")=0;
    i=0;
    k=0;
//Displaying number of people by the Captioner in the Monitor

    DoReactStr("CAM","M","CLEAR_SUBTITLES","title_id<L>"); //M - Number of Camera  L - Number of Captioner
    DoReactStr("CAM","M","ADD_SUBTITLES","command<Number of people (entering/exiting): "+i+" / "+k+"\r>,
page<BEGIN>,title_id<L>"); //M, L - the same
  }
```

As a result, when the corresponding camera is displayed in the Monitor, a text message about the number of entered and exited people will be superimposed on the video image.

> **Note**
>
> When you use the page<BEGIN> and page<END> parameters, the corresponding fields of the captions database are filled in. This enables data search using the **Captions search** interface object.

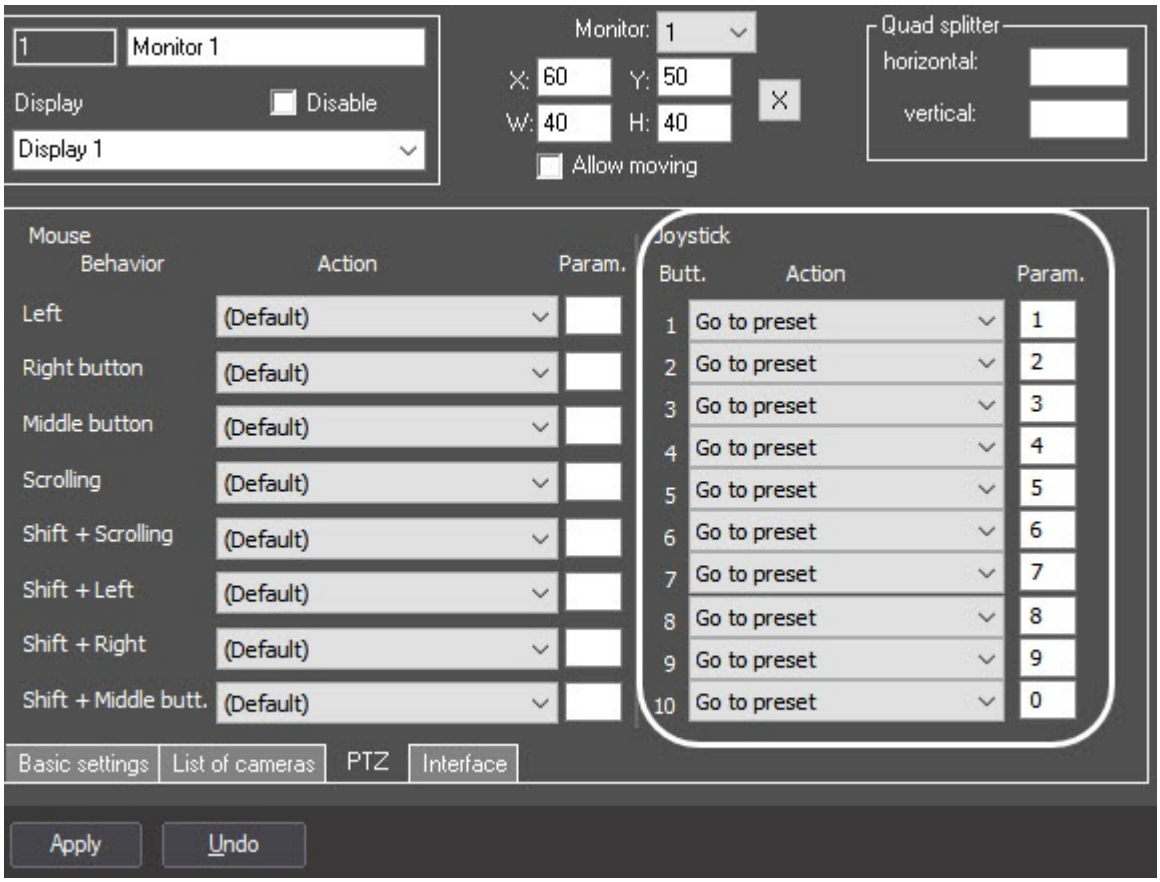
You can configure the font, color and position of text on the settings panel of the **Captioner** object (see Configuring captions display on a video image).

To null people counter, you must first create the **Macro** object on the **Programming** tab. You can change the name of the **Macro** object, for example "Null people counter".

You can run the macro to null people counter either manually from the main menu of *Axxon PSIM* or automatically at any specified time (for this, use the **Events** table on the settings panel of the **Macro** object on which you must specify the previously configured **Time schedule** object). For more information about the use of the **Macro** and **Time schedule** objects, see Administrator's Guide.

Example 3. Displaying camera on the monitor by clicking the button on the control panel

The following example is valid only for cameras in configuration of which there is a PTZ control panel. When configuring **Video surveillance monitor**, select the **Go to preset** action with 1,2,3...,0 parameters for ten joystick buttons (see Assigning commands to joystick buttons using the Monitor section of Installing and configuring security system components guide).



Example. When the button is clicked on the control panel, display the corresponding camera in the active Monitor. The script must be triggered by a timer with ID=1.

> **Note**
>
> You must create and configure the **Timer** object beforehand and set the current year. For detailed information on configuring the **Timer** object, see Creating and configuring the Timer object.

After each button click on the control panel wait for two seconds until clicking another button. If there is no button click, then the camera with dialed number must be displayed.

```
if (Event.SourceType=="TIMER" && Event.SourceId=="1" && Event.Action=="TRIGGER")
{
  mon="1";
  DebugLogString("on monitor "+ 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_PRESET")
{
  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"));
}
```

Example 4. Superimposing captions

On Macro 1, display the text

"NNN

Titles"

(with line break) over the video image of camera 1 using captioner 1. On Macro 2, disable the display of this text.

```
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>");
}
```

Examples of scripts with Map

MAP Map

Specifying the text to display on the map

When you add an object to the Map, you can select the type of the **Text** display (see Attaching objects to the layers of interactive map) . You can use the script in the JScript language to change the displayed text.

Example. The Text type of display on the map is selected for the Camera 1. It's required to display the value of the MyVar variable, read from the C:\test.ini file, on the map and debug window on Macro 1.

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

Examples of scripts with detection tools

CAM_VMDA_DETECTOR VMDA detection

CAM_IP_DETECTOR Embedded detection

Example 1. Script to select abandoned objects with a frame in live video

If you use the object tracking on a video image (see Creating and configuring the Tracker object), then when you view archive, the detected abandoned objects will be selected with a framed in video. To select the abandoned objects with a frame in a live video, use the script to select an abandoned object with a frame when receiving an alarm from VMDA detection tool:

```
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<"+x1+">,x2<"+x2+">,y1<"+y1+">,
y2<"+y2+">");
   DebugLogString("x1:"+x1+" x2:"+x2+" y1:"+y1+" y2:"+y2);
 }
 else
 {
   DoReactStr("MONITOR","","DEL_MARKRECT","cam<"+cam+">,id<"+cam+">");
 }
}
```

Example 2. Using the embedded People counter detection on Bosch FLEXIDOME IP dynamic 7000 VR IP camera

When the number of people reaches 20 on the embedded people counter detection of the Bosch FLEXIDOME IP dynamic 7000 VR IP camera (with ID 1), call macro 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","");
 }
}
```

Examples of scripts with Macros

MACRO Macro

Example 1. Sending a command to a camera using the camera HTTP API

Camera IP address is 192.168.0.13.

The following command turns on the screen wiper on a camera:

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

The following command turns off the screen wiper on a camera:

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

These commands must be sent to a camera using the JScript script.

```
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");
}
```

Example 2. Sending e-mail with HTML markup

On macro 1, send a message with the attached detected.png and found.jpg files from the C:\\Pictures\\ folder to example@gmail.com. The message must be formatted as follows:

Face detected

Detected face	Face in DB
detected.png file	found.jpg file

```
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<daniel@axxonsoft.com>,objname<Mail message 1>,subject<>,
parent_id<1>,flags<>,pack<>,name<Mail message 1>,from<example@gmail.com>,_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>E-mail sent from Axxon PSIM</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);
}
```

Example of script with Users

PERSON User

CORE

MACRO Macro

Creating test users

On macro 101, create 50 users in *Axxon PSIM* with IDs from 100 to 150, assigning them an access level with ID 1 (provided that the access level is assigned to the department to which the users are added and users inherit the department access level) and linking an access card with a number, equal to the user ID. The card number must be in HEX format. The department must have no more than 30 users (to speed up the adding process).

Note

For more information on access levels and access cards, see the *ACFA PSIM* documentation in the AxxonSoft documentation repository.

If an ACS integration is configured in *Axxon PSIM*, which supports dynamic user recording, then the created user will be automatically written to the ACS controller when the CORE||UPDATE_OBJECT|objtype<PERSON> event is sent. If dynamics is not supported, then recording users to the controller will need to be initiated manually.

```
dep=10; // department ID
start=100;// first user ID
last=150; // last user ID
acc_lev=1; // access level ID
dep_count=30; // max number of users in the department

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

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

Examples of scripts with Incident server and Incident manager

INC_SERVER Incident server

INC_MANAGER Incident manager

Example 1. On macro 1, change the status of the Alarm event on camera 1 to Completed.

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

Example 2. On macro 2, on Incident server 1, change the status of the event escalation of camera 1 to Waiting for processing (not escalated).

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

Example 2. Changing the status of an event in Incident manager

When working with objects in the **Incident manager,** it is possible to change the event status of an object (see Processing events). To change the event status of an object, you can use a JScript script.

Example. On macro 3, change the status of the Alarm event on camera 1 or 2 to Completed.

```
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>");
 };
```

Example of script with Failover service

FAILOVER Failover service

Example 1. Using the START and STOP events for the Failover service

Objects from more than one main Server must not be moved to the Backup Server. For this, when moving objects from some main Server to the Backup Server, all other **Failover service** objects must be disabled on this Backup Server.

```
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,"");
     }
  }
}
```

Examples of scripts with BacNet

BACNET BacNet

Example 1. Writing to an object using a script

```
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);
```

If the script is successfully executed, an event will appear in the **Debug window**:

```
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>
```

Example 2. Event generation

```
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);
```

If the module receives an event, the following event will appear in the **Debug window**:

```
Event:
BACNETINT|1|EVENT_OCCURES|sender<Udp:47808>,slave_id<ASUS>,fraction<683>,owner<ASUS>,
event_type<EVENT_CHANGE_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>
```

Example 3. Reading data from an object

```
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);
```

If the reading is successful, an event will appear in the **Debug window**:

```
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>
```

Example with Telegram bot

TELEGRAM Telegram bot

Sending a message to Telegram

On macro 1, send text, image or geolocation with Telegram bot.

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

{

  //Sending with chat_id & bot_id from object settings:

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

  //Explicit setting chat_id & bot_id in the command:

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

  //Sending file with chat ID and bot ID:

  DoReactStr("TELEGRAM",1,"SENDPHOTO","caption<Axxon PSIM 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>);

}
```

Examples of scripts with Event Viewer

EVENT_VIEWER Event Viewer

Activating one filter

Example. In the **Event Viewer**, activate Filter 1 for Event viewer 1 using Macro 1.

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

Activating multiple filters

Example. In the **Event Viewer**, activate Filter 1, Filter 3, and Filter 5 for Event viewer 1 using Macro 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>");
}
```

Appendix 1. Description of the Editor-Debugger utility

The purpose of the Editor-Debugger utility

The *Editor-Debugger* utility is used to create, debug and edit scripts in *Axxon PSIM*.

The *Editor-Debugger* utility provides the following functionality:

1. Creating and editing scripts using the built-in text editor.
2. Debugging scripts using the built-in debugging window.
3. Filtering the information to be displayed in the debugging window.
4. Creating and using test events for debugging.
5. Saving scripts to the hard drive;
6. Opening scripts from the hard drive.

The interface of the Editor-Debugger utility

The Editor-Debugger interface

The user interface of the *Editor-Debugger* utility contains the main menu and the toolbar (**1**), the objects list (**2**) and the viewing /editing panel (**3**).

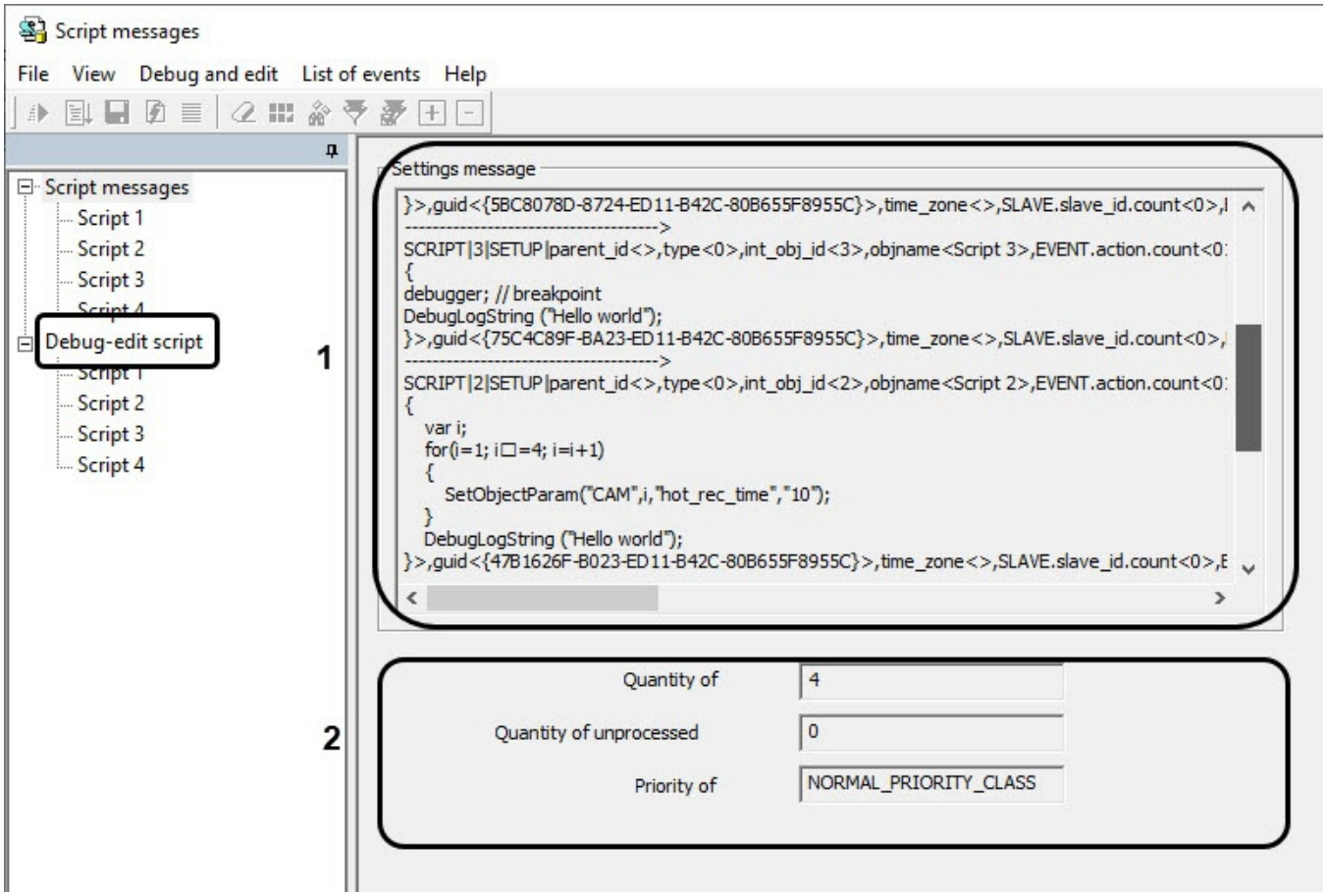
The Debug-edit script tab

On the page:

Description of the interface of the Debug-edit script tab

The **Debug-edit scrip** tab is used to edit scripts and create test events.

The figure below shows the interface of the **Debug-edit scrip** tab:



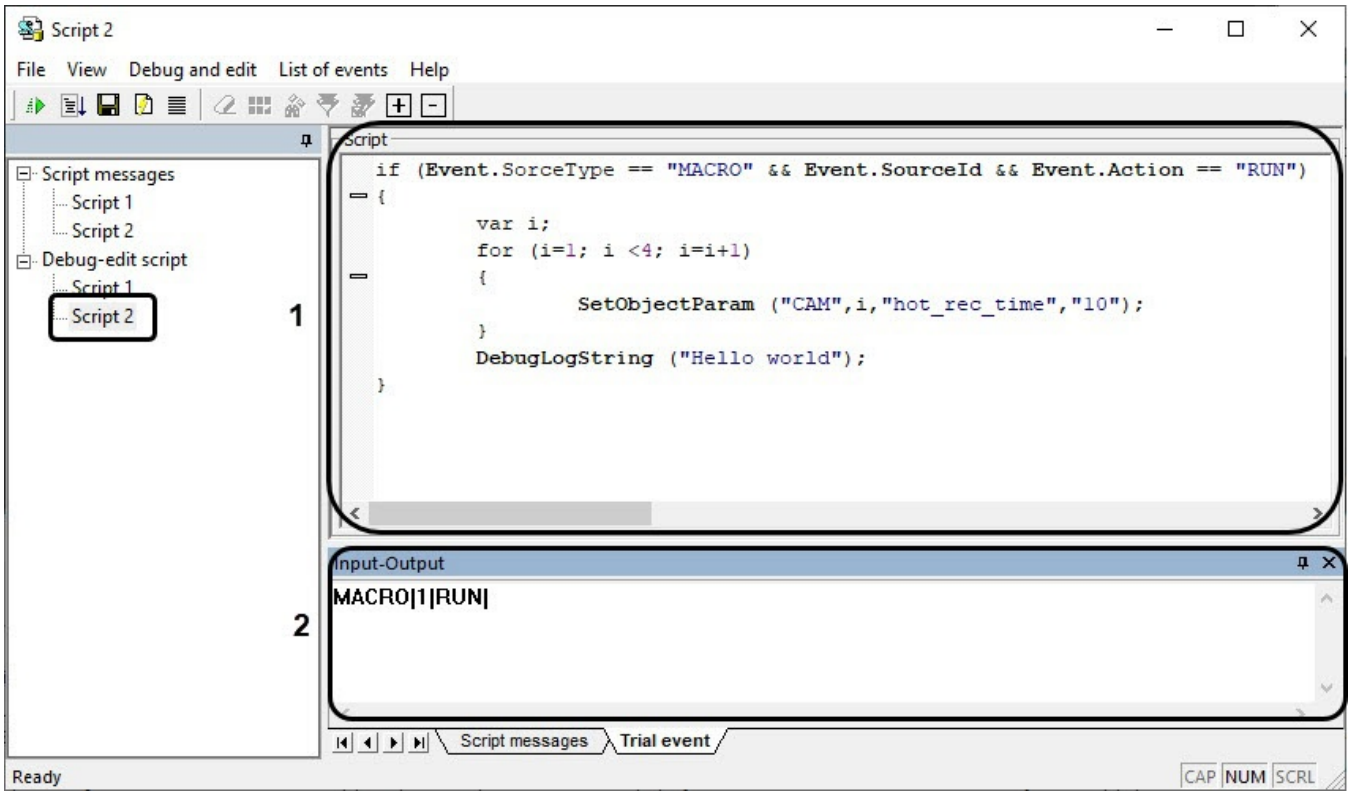
Description of the interface of the **Debug-edit scrip** tab:

Number in the image	Parameter name	Method of setting the parameter value	Parameter description
1	Settings message	Automatically	Information about initialization of the **Script** objects in the system
2	Additional information	Automatically	Additional information about scripts

Description of the Script interface object (the Debug-edit scrip tab)

The **Script** object in the **Debug-edit scrip** tab is used to creat and edit scripts and test events.

The figure below shows the interface of the **Script** object:



Description of the interface of the **Script** object:

Number in the image	Parameter name	Method of setting the parameter value	Parameter description	Characters used	Default value	Value range
1	Script	Enter the value into the field	Contains the text of the script	Latin, Cyrillic and service characters	Blank line	Unlimited number of characters
2	Event	Enter the value into the field	Contains the text of the test event	Latin, Cyrillic and service characters	Blank line	Unlimited number of characters

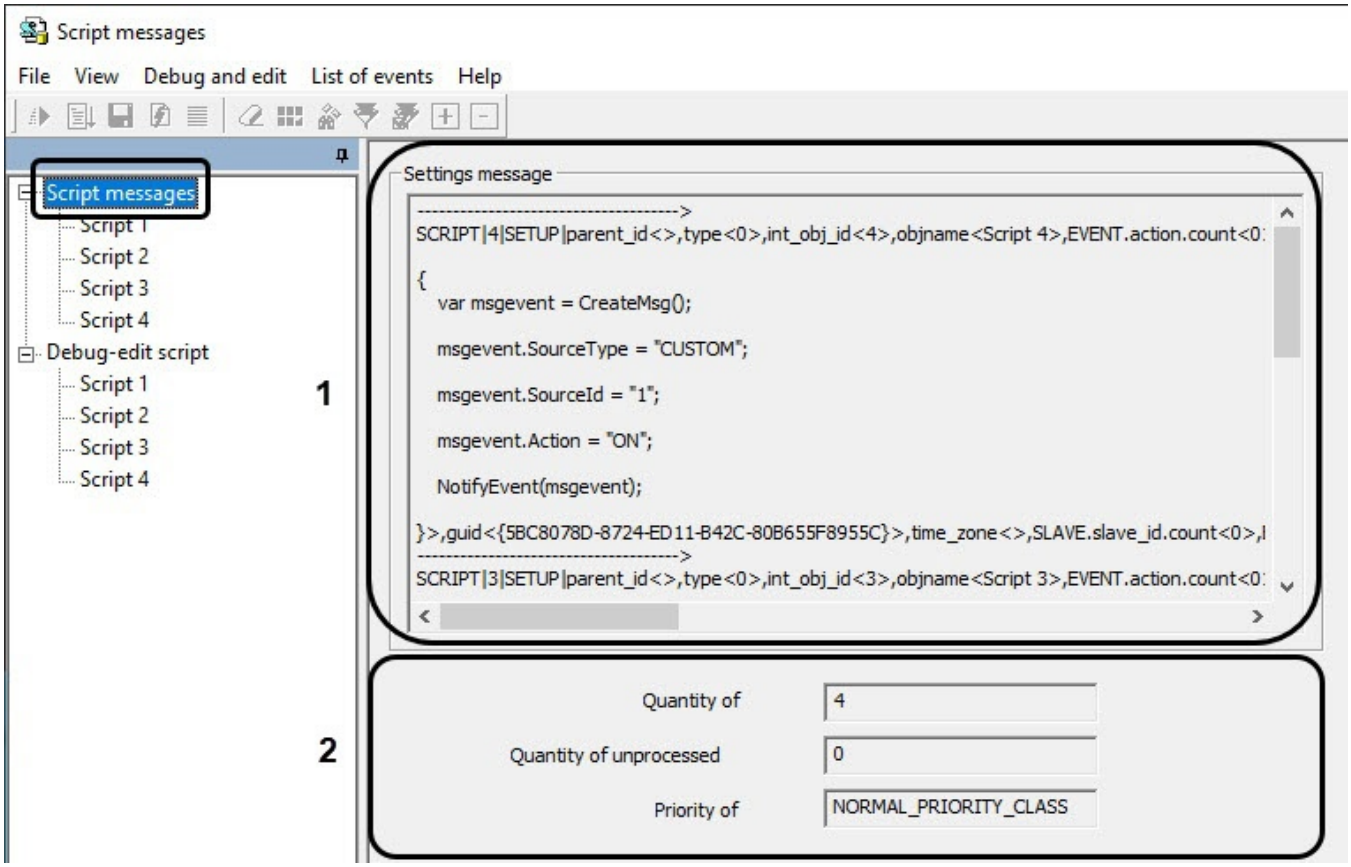
The Script messages tab

On the page:

Description of the interface of the Script messages tab

The **Script messages** tab is used to display the debugging windows of scripts.

The figure below shows the interface of the **Script messages** tab:



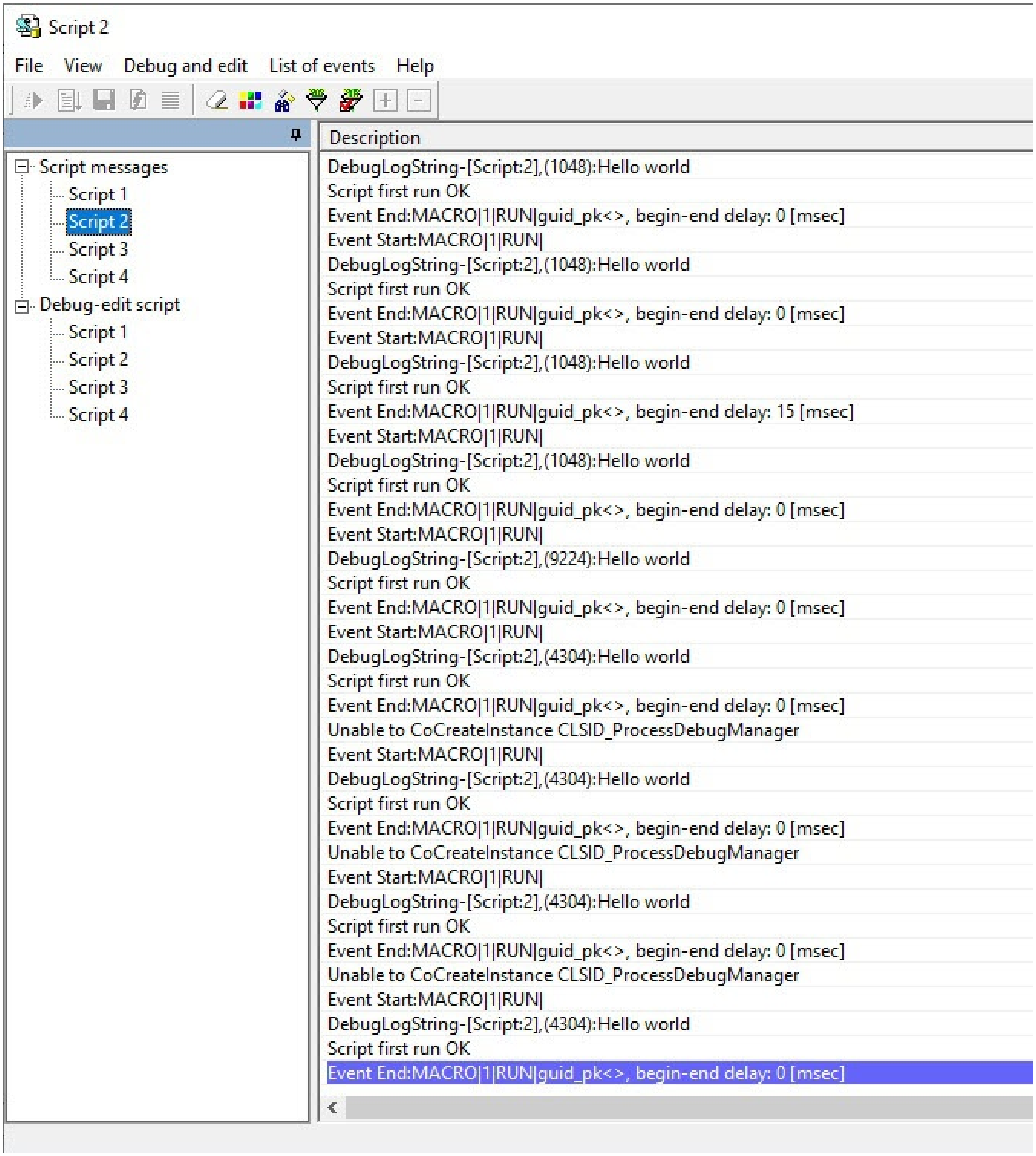
Description of the interface of the **Script messages** tab:

Number in the image	Parameter name	Method of setting the parameter value	Parameter description
1	Settings message	Automatically	Information about initialization of the **Script** objects in the system
2	Additional information	Automatically	Additional information about scripts

Description of the Script interface object (the Script messages tab)

The **Script** object in the **Script messages** tab is used to to display system, test and user events related to the scripts created in *Axxon PSIM*.

The figure below shows the interface of the **Script** object:



Description of the interface of the **Script** object:

Number in the image	Parameter name	Method of setting the parameter value	Parameter description	Characters used	Default value	Value range
1	Description	Automatically	Displays information about events occurring in the system	Latin, Cyrillic and service characters	Undefined	By default 200 lines are displayed in the list. You can change the value using the DebugMaxLines registry key (see Registry keys reference guide)

Main menu

Description of the main menu interface

The main menu of the *Editor-Debugger* utility is used to call the commands executed by the utility. The commands are divided into groups by functional attributes and are located in the following menu items: **File**, **View**, **Debug and edit**, **List of events** and **Help**.

Description of the items of the main menu:

№	Parameter name	Method of setting the parameter value	Parameter description
1	File	Select the command from the drop-down list	Contains a list of commands for saving and opening scripts, closing the utility
2	View		Contains a list of commands for displaying the toolbar and status bar in the utility window
3	Debug and edit		Contains a list of commands for script debugging
4	List of events		Contains a list of commands changing the parameters of message display in the debugging windows
5	Help		Contains the **About** command. This command displays the window with general information about the *Editor-Debugger* utility

Description of the File item of the main menu

The **File** item of the main menu is used to open and save scripts and to close the utility.

Description of the elements of the **File** item of the main menu:

№	Parameter name	Parameter description
1	Save to database	Saves the script to its corresponding system object
2	Save to disk	Saves the script to a text file on the hard disk
3	Open from disk	Opens the script from the selected text file into the utility editor
4	Exit	The command shuts down the utility and closes its dialog window

Description of the View item of the main menu

The **View** item of the main menu is used to call the commands that enable and disable the display of the toolbar and status bar in the utility window.

Description of the elements of the **View** item of the main menu.

№	Parameter name	Method of setting the parameter value	Parameter description	Default value
1	Toolbar	Set the checkbox	Enables or disables the display of the toolbar in the utility window	Checkbox is set
2	Status bar	Set the checkbox	Enables or disables the display of the status bar at the bottom of the utility window	Checkbox is set

Description of the Debug and edit item of the main menu

The **Debug and edit** item of the main menu is used to call the commands for debugging scripts.

Description of the elements of the **Debug and edit** item of the main menu:

№	Parameter name	Parameter description
1	Test run	Runs the script on a test event
2	Test run in third-party d ebugger	Runs the script on a test event using the third-party debugger
3	Edit test event	Opens the dialog window for editing a test event
4	Summary information	Opens the **Thread Information** window, which displays the system, test and user messages related to the script being debugged
5	Go to line	Opens the dialog window for specifying the number of the character and line in the script text to which you want to go

Description of the List of events item of the main menu

The **List of events** item of the main menu is used to change the parameters of the message display in the debugging window.

Descriptions of the elements of the **List of events** item of the main menu:

№	Parameter name	Parameter description
1	Clear	Clears the **Description** field of the debugging window
2	Highlight	Opens the window used to specify words (or other character sequences) in strings containing them, which you want to highlight in the **Description** field of the debugging window
3	Search	Searches for a word (or other character sequences) in the **Description** field of the debugging window
4	Set filter	Opens the window used to enable and configure a filter. The strings that contain (don't contain) the specified words must be included in the debugging window (or excluded from it)
5	Apply filter	Applies the created filter

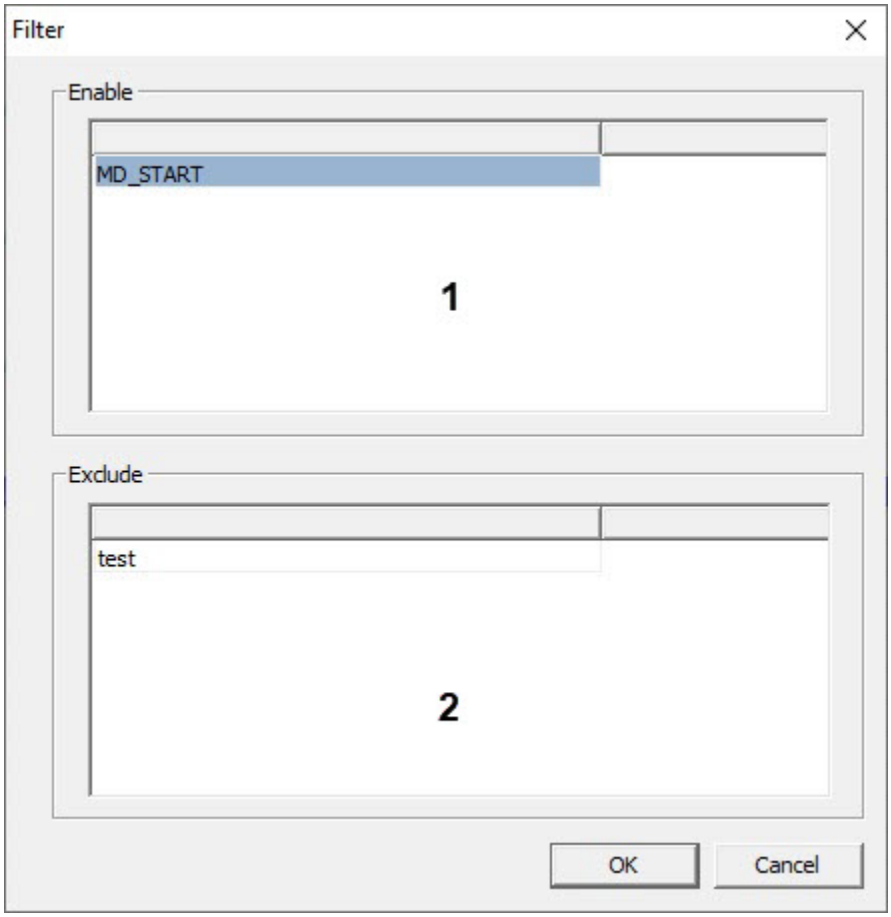
Description of the Filter dialog window

The **Filter** dialog window is used to enable and configure message filters displayed in the **Description** field of the debugging window.

You can open the **Filter** interface window in two ways:

1. Click the **Filter**  button in the toolbar of the *Editor-Debugger* utility.
2. In the **List of events**, select **Filter**.

Interface of the **Filter** window:



Description of the interface of the **Filter** window:

Number in the image	Parameter name	Method of setting the parameter value	Parameter description	Characters used	Default value	Value range
1	Include	Enter the value into the text field	The strings containing the words (or other character sequences) specified in this field will be displayed in the debugging window	Latin, Cyrillic and service characters	Blank string	Unlimited number of characters
2	Exclude	Enter the value into the text field	The strings containing the words (or other character sequences) specified in this field will be excluded from the debugging window	Latin, Cyrillic and service characters	Blank string	Unlimited number of characters

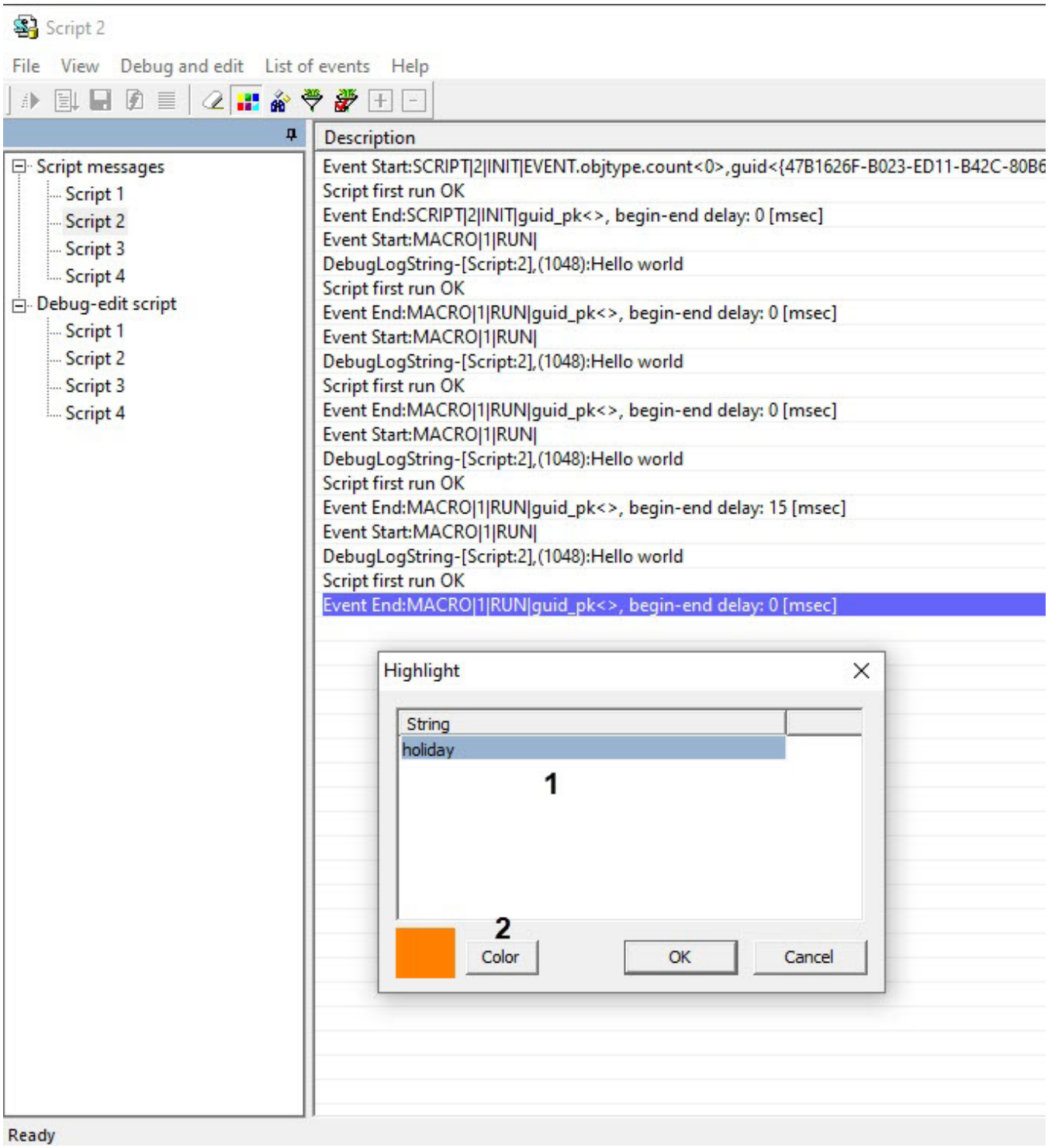
Description of the Highlight dialog window

The **Highlight** dialog window is used to set the color highlighting of the lines containing specified words in the debugging window.

You can open the **Highlight** dialog window in two ways:

1. Click the **Highlight** button in the toolbar of the *Editor-Debugger* utility.
2. In the **List of events** menu, select **Highlight**.

Interface of the **Highlight** window:



Description of the elements of the **Highlight** window:

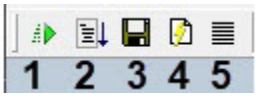
Number in the image	Parameter name	Method of setting the parameter value	Parameter description	Units	Default value	Value range
1	String	Enter the value into the text field	Sets the words (or other character sequences) to highlight the strings containing them in the debugging window	Latin, Cyrillic and service characters	Blank string	Unlimited number of characters
2	Color	Select the value from the drop-down list	Sets the color to highlight strings in the debugging window	RGB format	Gray	RGB colors

Description of the toolbar of the Editor-Debugger utility

The toolbar of the *Editor-Debugger* utility is used to call the frequently used functions of the utility.

The toolbar operates in two modes: when the script control buttons are active, or when the debugging window control buttons are active. The mode depends on the currently active tab of the *Editor-Debugger* utility: either the **Debug-edit script** tab for editing scripts, or the **Script messages** tab for viewing messages in the debugging window.

Interface of the toolbar of the *Editor-Debugger* utility in the script editing mode:



Description of the interface of the toolbar of the *Editor-Debugger* utility in the script editing mode:

Number in the image	Parameter name	Parameter description
1	Test run	Runs the script on a test event
2	Test run in third-party debugger	Runs the script on a test event using the third-party debugger
3	Save	Saves the script to the **Script** system object
4	Edit test event	Opens the dialog window for editing test events
5	Summary information	Opens the **Thread Information** window, which displays the system, test and user messages related to the script being debugged

Interface of the toolbar of the *Editor-Debugger* utility when working with the debugging window:

Description of the interface of the toolbar of the *Editor-Debugger* utility when working with the debugging window:

Number in the image	Parameter name	Parameter description
1	Clear	Clears the **Description** field of the debugging window
2	Highlight	Opens the window used to specify words (or other character sequences) in strings containing them, which you want to highlight in the **Description** field of the debugging window
3	Search	Searches for a word (or other character sequences) in the **Description** field of the debugging window
4	Set filter	Opens the window used to enable and configure a filter. The strings that contain (don't contain) the specified words must be included in the debugging window (or excluded from it)
5	Apply filter	Applies the created filter

Appendix 2. Creating custom objects with ability to set events, reactions and states

Purpose of custom objects and their implementation in Axxon PSIM

Custom objects represent software emulation of new *Axxon PSIM* objects and allow for configuring their states, reactions, and events. You can work with custom objects using scripts, macros and macroevents.

You can create custom objects using the ddi.exe and CustomTypeEditor.exe utilities in the <*Axxon PSIM* installation folder>\Tools.

In the How to create a custom object section, you can find an example of creating two types of custom objects that can be used to show the state of the abandoned objects detection tool on the map or to show any other user states. The states of objects are changed using macros, scripts, or via IIDK.

The procedure for creating and configuring a custom object:

1. Prepare the DBI file with the required types of objects.
2. Prepare the DDI file—it specifies the events, reactions, states and state transition rules for the newly created objects.
3. Prepare the XML file with the parameters of the newly created objects.
4. Update the main database using The idb.exe utility for converting databases, selecting database templates and making backup copies of databases.
5. Create a custom object in *Axxon PSIM*.

How to create a custom object

Here you can find out how to create the following custom objects:

1. CUSTOM type with SLAVE (Computer) parent type.
2. CUSTOM_CHILD type with CUSTOM parent type (see item 1).

An object of the CUSTOM type has the property set:

1. Custom_param1 and custom_param2 parameters
2. Events: ALARM, INFO, ON, OFF
3. Reactions: ON, OFF
4. States: ON, OFF
5. State machine:
    a. Set ON state for ON event
    b. Set OFF state for OFF event

The CUSTOM_CHILD type of the object is created to demonstrate tree structure and has no user parameters, events, reactions or states.

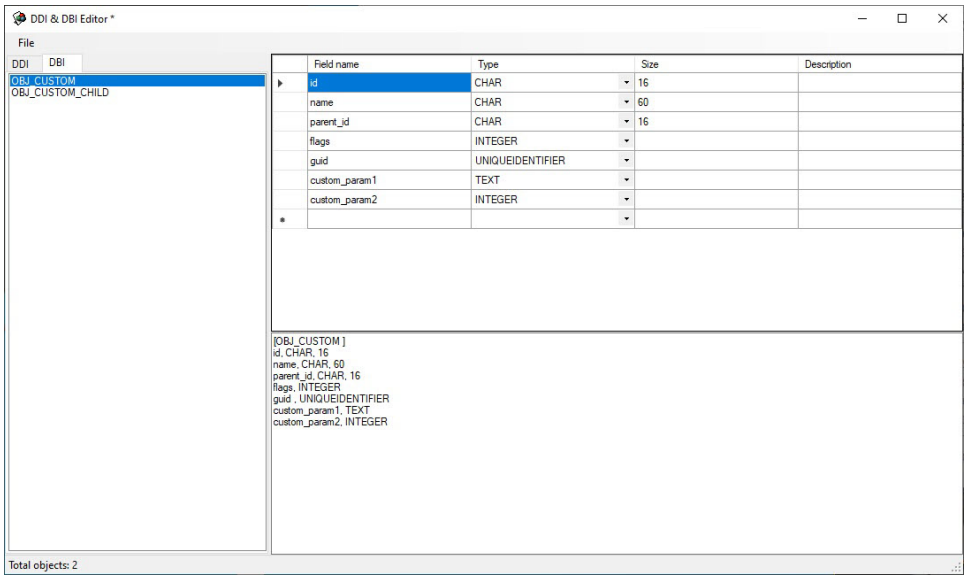
DBI file preparation

A DBI file is prepared using the ddi.exe utility. Details on how to handle it can be found in The ddi.exe utility for editing database templates and external settings files.

A DBI file for the objects of the CUSTOM and CUSTOM_CHILD type is created as follows:

1. Run the dbi.exe utility (see The ddi.exe utility for editing database templates and external settings files).
2. Go to the **DBI** tab.
3. Create two objects—OBJ_CUSTOM and OBJ_CUSTOM_CHILD as shown in the figure below.

> **Attention!**
>
> Object (table) names must look like OBJ_<object type>.



4. Set the parameters for each object. The **id**, **name**, **parent_id**, **flags**, **guid** parameters are mandatory for all objects. **Custom _param1**, **custom_param2** in the example in the figure are custom parameters. You can also set other parameters used in *Ax xon PSIM.* For example, adding the **region_id** parameter will allow you to set areas and regions for an object (see Subdivision of the protected facility into areas and regions).
5. Save the changes using the **Save** command in the **File** menu. The saved file must have the dbi extension and must be located in *Axxon PSIM* installation directory, for example, C:\Program Files (x86)\Axxon PSIM\PSIM.custom.dbi.

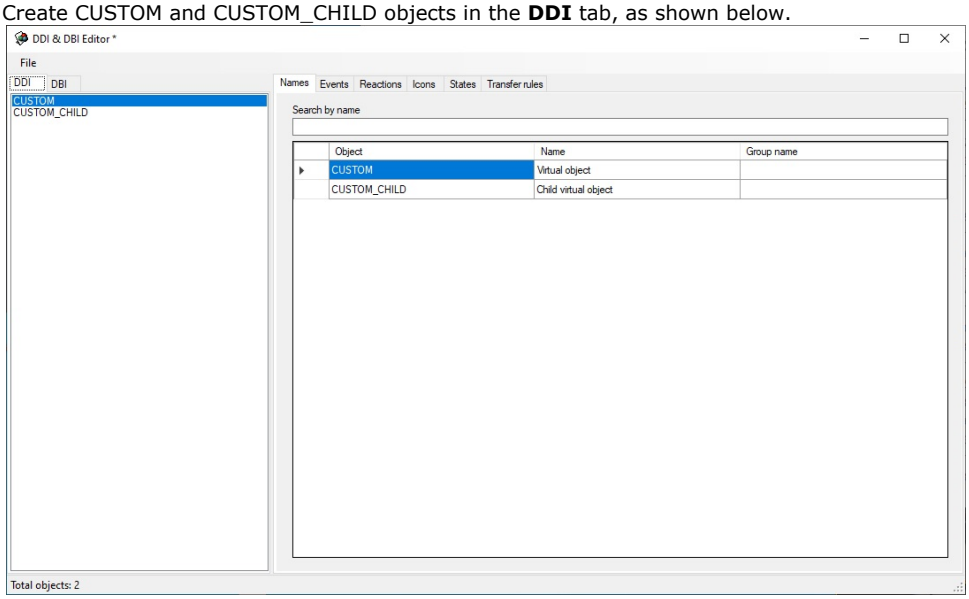
DBI file preparation is complete.

DDI file preparation

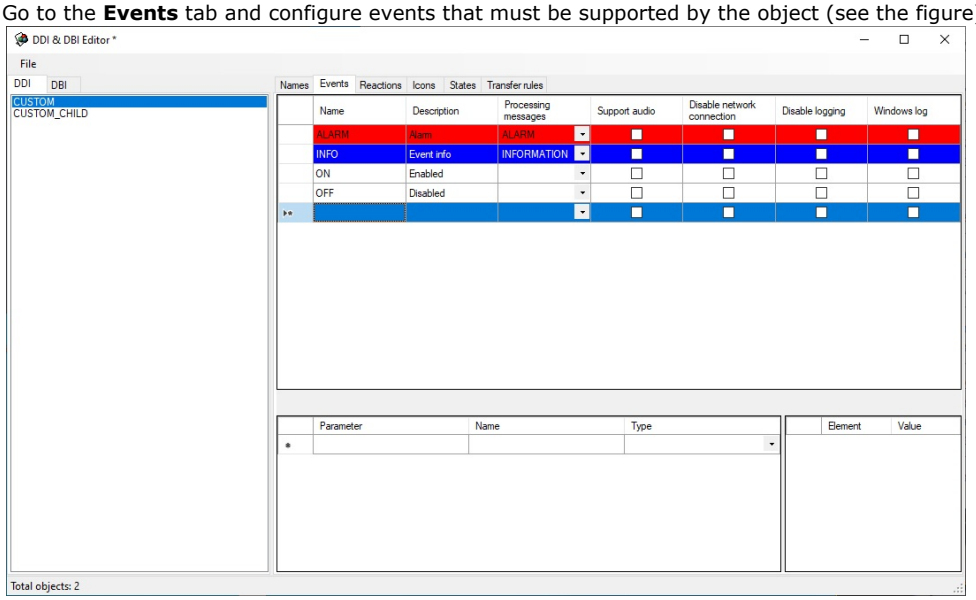
A DDI file is prepared using the ddi.exe utility. Details on how to handle it can be found in The ddi.exe utility for editing database templates and external settings files.

A DDI file for CUSTOM and CUSTOM_CHILD object types is created as follows:

1. Run the ddi.exe utility (see The ddi.exe utility for editing database templates and external settings files).
2. Create CUSTOM and CUSTOM_CHILD objects in the **DDI** tab, as shown below.

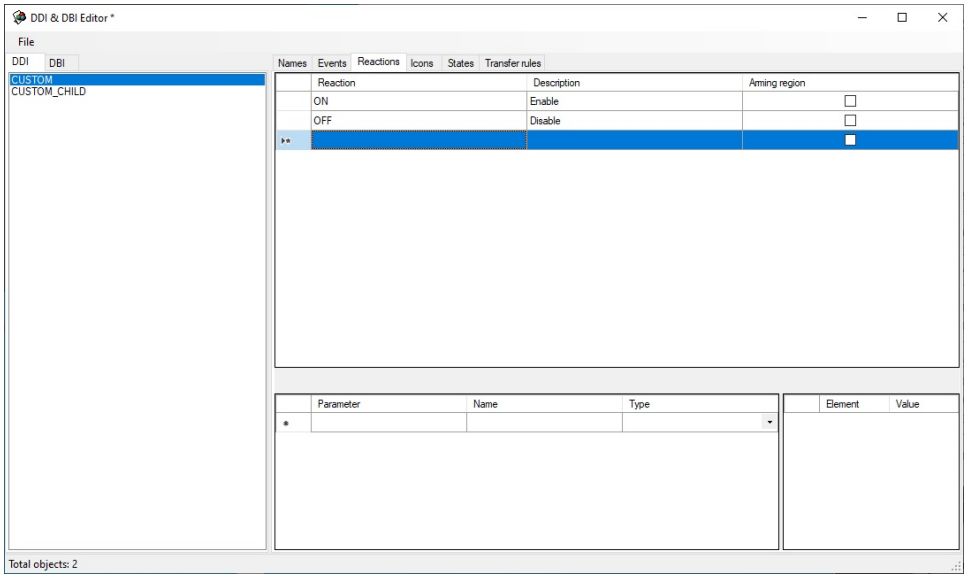


3. Go to the **Events** tab and configure events that must be supported by the object (see the figure).

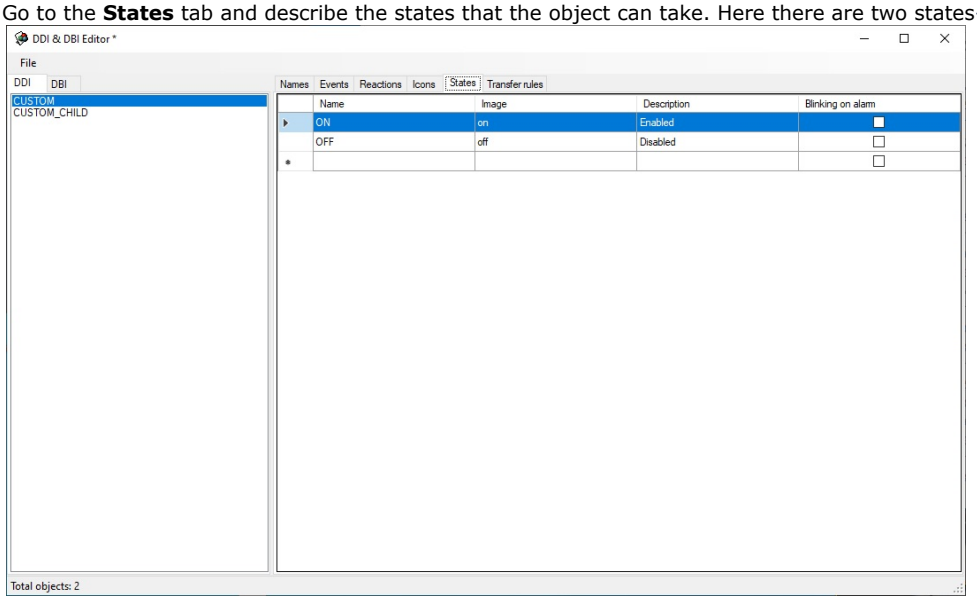


4. Go to the **Reactions** tab and configure reactions that must be supported by the object (see the figure).

> **Note**
>
> Reactions of custom objects are automatically converted into events. In other words, a custom object automatically generates an event when there is a reaction.

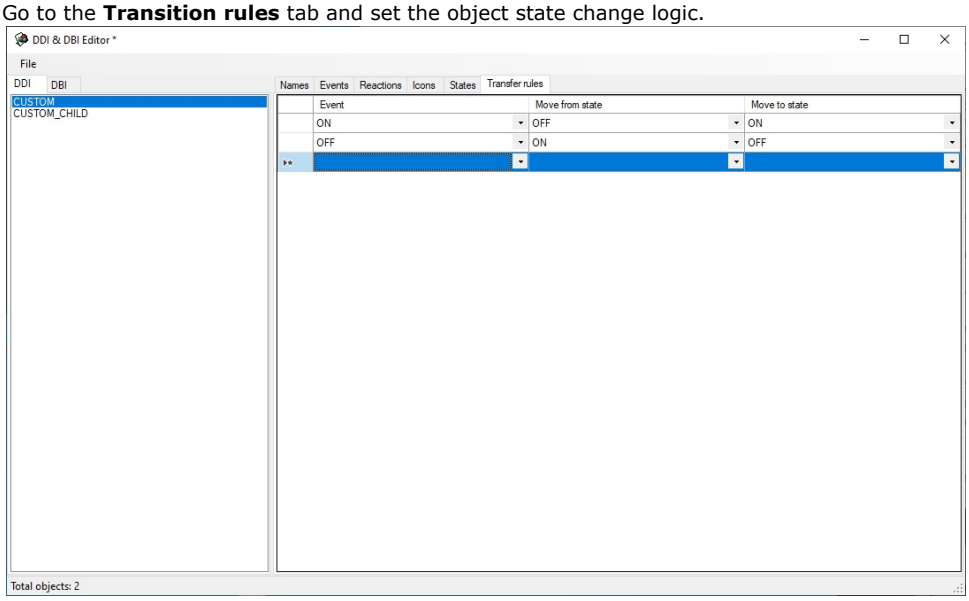


5. Go to the **States** tab and describe the states that the object can take. Here there are two states—ON and OFF.



> **Note**
>
> The postfix of file name is specified in the **Image** column—the image that is stored in <*Axxon PSIM* installation directory>\Bmp. For instance, these will be custom_off.bmp and custom_on.bmp files (corresponding to ON and OFF states) for CUSTOM object. These files will be used by the map module.

6. Go to the **Transition rules** tab and set the object state change logic.

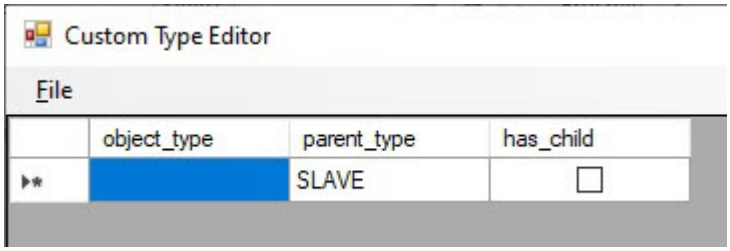


   Transition rules is a simple state machine—an event is an input action and a state is a result.
   An unconditional transition is used in this case: if CUSTOM||ON event is received, then there is transition to the ON state, if CUSTOM||OFF event is received, then there is transition to the OFF state.
7. To save changes use the **Save** command in the **File** menu. The saved file must have the ddi extension and be stored in the folder corresponding to the required language, for example, C:\Program Files (x86)\Axxon PSIM\Languages\en\psim.custom.ddi.

The DDI file preparation is completed.

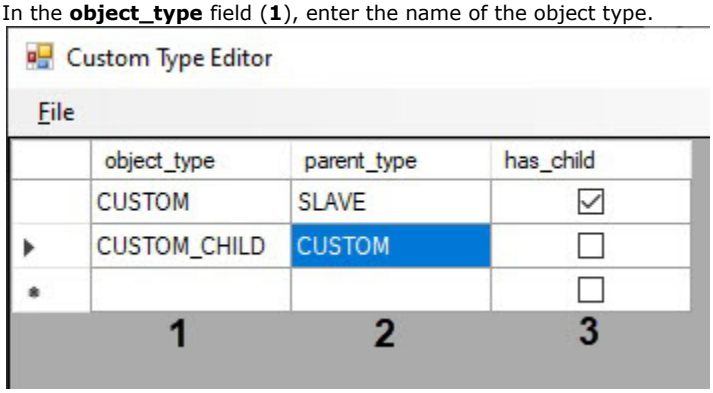
XML file preparation

An XML file is prepared using the CustomTypeEditor.exe utility that can be found in <*Axxon PSIM* installation directory>\Tools. The general view of the utility window is shown in the figure below.



An XML file for a custom object is created as follows:

1. In the **object_type** field (**1**), enter the name of the object type.



2. In the **parent_type** field (**2**), enter the name of the parent type.
3. If the object type has child types, then set the **has_child** checkbox (**3**).
4. Repeat steps 1-3 for all object types.
5. Save the file with any name and .xml extension in the *Axxon PSIM* installation directory using the **File Save** command. For example, the "CUSTOM.xml" file name is recommended for the object shown on the picture above.

The XML file is now created. The file contents look like this:

```
<?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>
```

You can edit it manually if required.

In particular, the <include_parent_id>1</include_parent_id> parameter can be added to an XML file. When setting this parameter to 1, the IDs of the child custom objects will include the ID of the parent object. For example, if a CUSTOM object has a child CUSTOM_CHILD, and the CUSTOM has ID = 3, then CUSTOM_CHILD objects will be created with identifiers 3.1, 3.2, and so on.

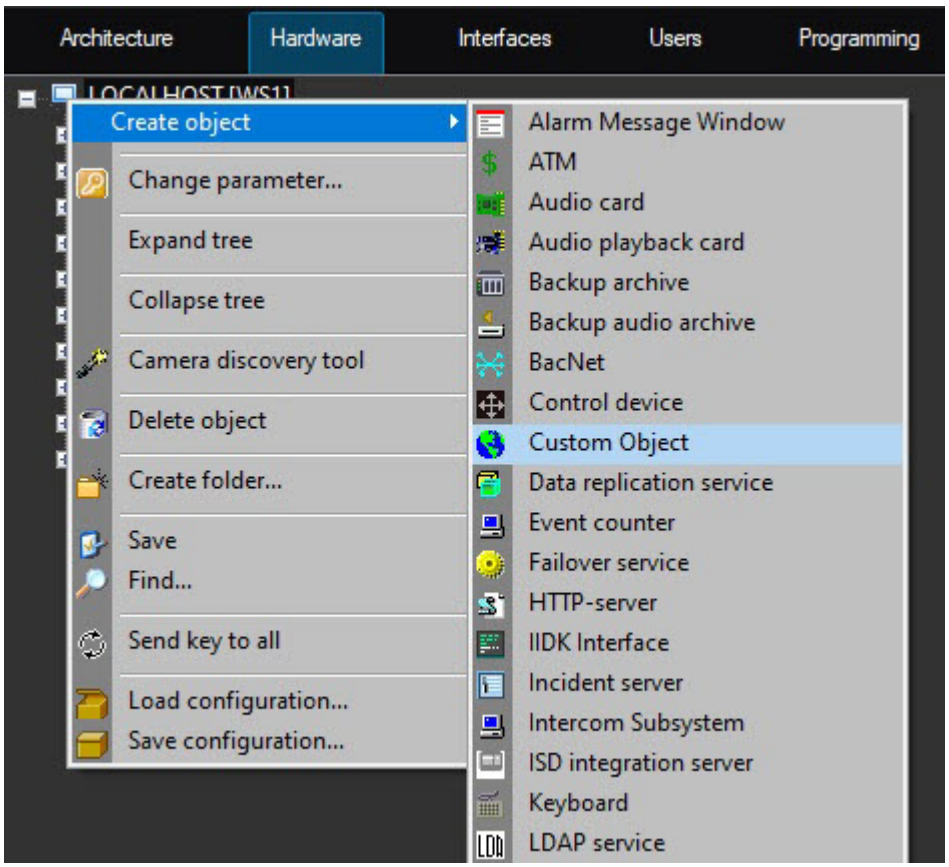
Creating and using a custom object in Axxon PSIM

On the page:

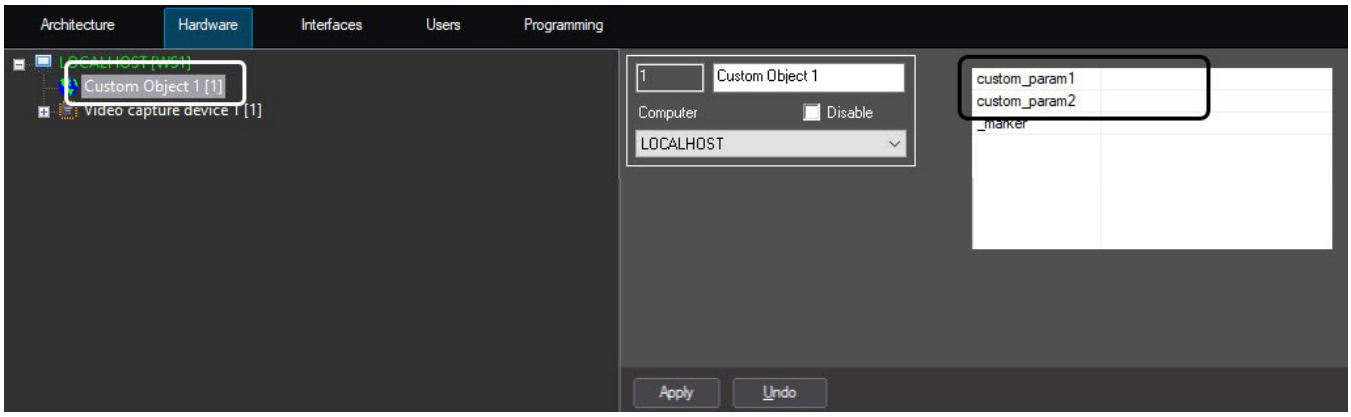
Attention!

After preparing the required files and before creating custom objects in *Axxon PSIM,* it is necessary to update the main database using The idb.exe utility for converting databases, selecting database templates and making backup copies of databases.

When the DBI, DDI and XML files are ready, the objects of a new type along with the standard objects can be created in *Axxon PSIM* hardware tree.

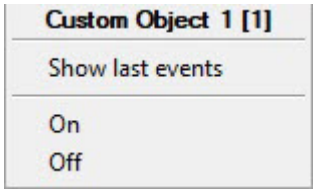


Custom parameters are displayed on the settings panel of the created custom object—custom_param1 and custom_param2 in this example. Their values can be set in the table.



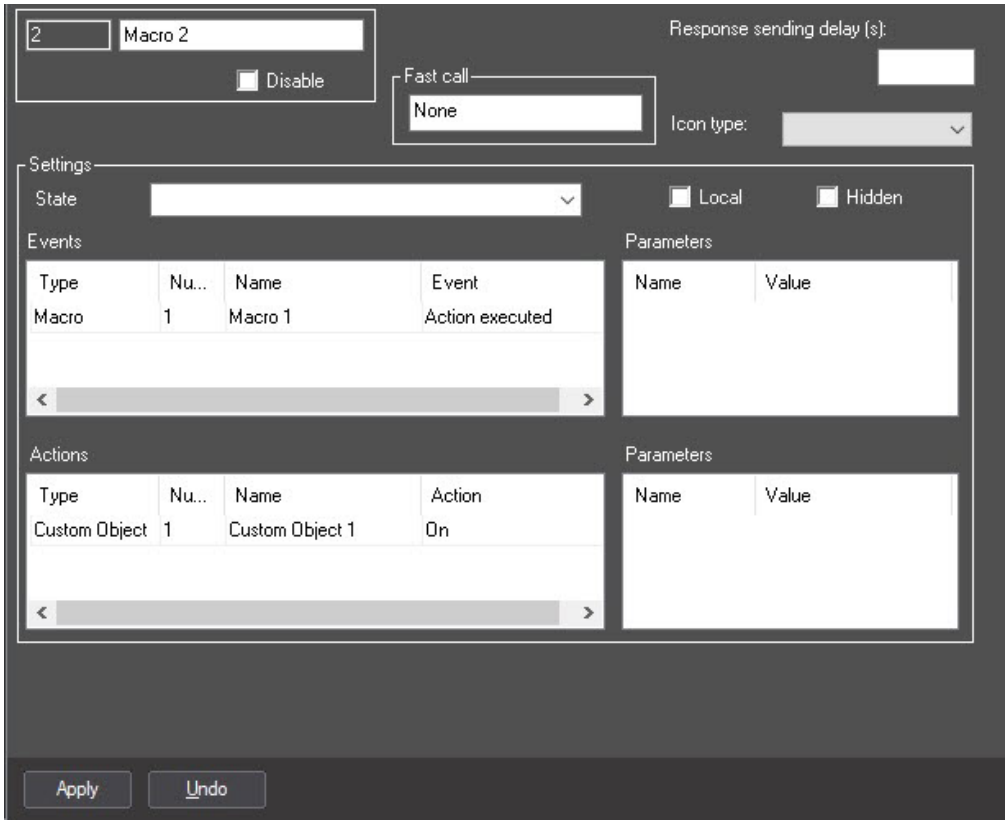
Displaying on the map

When an object is created in the hardware tree it can be placed on the map and set reactions can be executed in the object context menu (see Configuring the interactive map for object state indication and controlling the objects).



Using in macros

When a custom object is created in the hardware tree it can be used in macros.



> **Note**
>
> Reactions of custom objects are automatically converted into events. Thus, in the example, when the **ON** reaction is executed, the object state changes due to set state transition rules (see DDI file preparation) and the icon corresponding to the state will be displayed on the map.

Sample program in JScript to change the state of a custom object

Problem. Using macro 1 change the state of a custom object 1 to ON and display the icon corresponding to this state on the map.

Solution. As state transition rules are set, when the ON event is sent from the custom object, the state will be automatically changed to ON and the icon specified in ddi file (see DDI file preparation) to this state will be displayed on the map. A script for sending the ON event looks like this:

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

{

 var msgevent = CreateMsg();

 msgevent.SourceType = "CUSTOM";

 msgevent.SourceId = "1";

 msgevent.Action = "ON";

 NotifyEvent(msgevent);

}
```

Description of events and reactions of system objects

All reactions for the main objects of the system are specified in this section.

In *Axxon PSIM*, there are two types of system messages: events and reactions. Events are generally used for informational purposes—to send notifications to all *Axxon PSIM* cores that were connected to each other during architecture configuration. Reactions are commands sent to specific system objects. Reactions are sent directly to those cores on which the object is registered, and not to the entire system. To generate reactions, the DoReactStr and DoReact methods are used. To generate events, the NotifyEventStr and NotifyEvent methods are used.

Note

You can view events for system objects in one of the following ways:

1. Viewing the psim.ddi file using the ddi.exe utility (see Getting the list of system names of objects, reactions and events in Axxon PSIM).
2. Viewing events for the selected system object using the settings panel of the **Macro** system object (see Creating and configuring Macro events).

GRABBER Video capture device

The **Grabber** object corresponds to the **Video capture device** system object.

The **Grabber** object sends events presented in the table. Procedure is started when the corresponding event occurs.

Description of events of the **Grabber** object.

Events	Description
+12V	Voltage error +12V
+3.3V	Voltage error +3.3V
+5V	Voltage error +5V
-12V	Voltage error -12V
-5V	Voltage error -5V
CPU_FAN	Number of fan rotations
CPU_TEMP	Temperature of processor
SYS_TEMP	Temperature of MB chipset
UPS_COMMLOST	Connection lost
UPS_FATAL_ERROR	Error of connection
UPS_LOWBATT	Battery low
UPS_ONBATT	Switch to battery supply
UPS_ONLINE	Restoring the main supply
UPS_REPLACEBATT	Battery changing is required
UPS_SHUTTING	Shutdown
VCORE	Voltage of processor core
AUDIO_SIG_LOST	Sound lost
CONNECT_FAIL	Connection error
CONNECT_OK	Connected
NETWORK_FAILURE	Connection lost
STATE_CONNECTED	Connection restored

List of commands and parameters for the **Grabber** object is presented in the following table:

Command—command description	Parameters	Description of parameters
"SETUP"—sets parameters of video capture device	chan<>	Number of PCI slot (0,1,2,…,32).
	mode <>	Speed of grabber/digitising (0—maximal, 1—average, 2—minimal)
	resolution <>	Resolution (0—standard, frame quarter (384x288); 1—high, half-frame (768x288); 2—maximum, frame (768x576))

	format<>	Format of video signal (PAL, NTSC)
	drives<>	Disks for video archive record (DRIVE1:\, DRIVE2:\ … DRIVEN:\)
	cams<>	Number of connected video cameras
	auth<>	Authorization data
	ip<>	IP address of network video input card
	name<>	Name of object
	flags<>	Flags
	ip_port<>	IP port
	password<>	Password
	type<>	Type of digitising
	username<>	Login
	watchdog<>	WatchDog shutdown (0—disabled, 1—enabled)
"SET_DRIVES"—sets disks for video archive record	drives<>	Disks for video archive record
"MUX1_OFF"—disables video output through the analog output 1	-	-
"MUX2_OFF"—disables video output through the analog output 2	-	-
"MUX3_OFF"—disables video output through the analog output 3	-	-
"SET_IPINT_PARAM"—sets (changes) parameters of an IP device. Reaction allows changing of IP device settings wothout doing into its web interface. *Note. For the reaction to operate you must enable the multistream video—Configuration of multistream video, and Appendix 2. Defining the param_id and param_value values for the SET_IPINT_PARAM reaction*	param_id<>	Name of parameter. Set of parameters for each camera is individual—see Appendix 2. Defining the param_id and param_value values for the SET_IPINT_PARAM reaction
	param_value<>	Value of parameter. Set of parameters for each camera is individual—see Appendix 2. Defining the param_id and param_value values for the SET_IPINT_PARAM reaction
	cam_id<>	Camera ID in *Axxon PSIM*
	vstream_id<>	Number of video stream (optional parameter). It looks like “Number of camera”.”Number of stream”, for example, 1.1, 1.2

"START"—start playing video file in a virtual video capture device	-	-
"STOP"—stop playing video file in a virtual video capture device	-	-
"ENABLE"—enable object (clear the **Disable** checkbox in the object settings panel)	recursive<>	Possible parameter values: 0—enable only **Video capture device** 1—enable the **Video capture device** and all **Camera** objects created on the basis of it
"DISABLE"—disable object (clear the **Disable** checkbox in the object settings panel)	recursive<>	Possible parameter values: 0—disable only **Video capture device** 1—disable the **Video capture device** and all **Camera** objects created on the basis of it

Properties of the **GRABBER** object are shown in the table.

Properties of the GRABBER object	**Description of properties**
ID<>	Object ID
PARENT_ID<>	Number of video capture device

CAM Camera

The **CAM** object corresponds to the **Camera** system object.

The **CAM** object sends the events given in the table. Procedure is started when the corresponding event occurs.

Description of the events from the **CAM** object.

Events	Description	Comment
ARM	Camera is armed	If arming was performed by the Operator from the **Map** or **Video surveillance monitor**, the user_id<> parameter contains the identifier of the user who performed this action
ATTACH	Connecting	
BLINDING	Camera is sealed	
DETACH	Break	Event is generated when the input signal from the camera on the video capture device is lost
DISARM	Camera is disarmed	If disarming was performed by the Operator from the **Map** or **Video surveillance monitor**, the user_id<> parameter contains the identifier of the user who performed this action
FILE_REC_ERROR	Error of recording on disk	Event is generated when an error occurs when writing the video archive to disk
MD_START	Alarm	
MD_STOP	End of alarm	
PRINT	Print frame	
REC	Recording on disk	If recording was initiated by the Operator from the **Map** or **Video surveillance monitor**, the user_id<> parameter contains the identifier of the user who performed this action
REC_STOP	Stop recording on disk	If recording was stopped by the Operator from the **Map** or **Video surveillance monitor**, the user_id<> parameter contains the identifier of the user who performed this action
UNBLINDING	Camera is opened	
RECORDER_ON	Record is enabled	
RECORDER_OFF	Record is disabled	

DISC_MOUNT	Disk is mounted	
DISC_UNMOUNT	Disk is unmounted	
FINISHED_AVI_EXPORT	Video export is completed	If the video export fails, the event has the non-null error_result parameter. In the param<0> parameter there is additional information displayed in the corresponding column of the **Event Viewer**, in the following format: "ComputerName;ExportPeriod;UserName;UserID", for example, param0<LOCALHOST;04-06-18 16:50:55_04-06-18 16:55:55;Smith;1>
MD_LIMIT	Object count in a frame exceeded	The event is generated when the number of objects detected by the tracker in the frame is exceeded (see Creating and configuring the Tracker object section of the Administrator's Guide for details on how to set this parameter). An event is generated whenever the number of objects is changed (up or down), until it is less than the threshold. The object_count<> parameter is the number of objects in the frame that the tracker detected, exceeds the specified limit, and differs from the previous value. See also the description of the NEW_OBJECT event below
NEW_OBJECT	New object on a frame detected by tracker	Among others, it contains the following parameters: • total<> is the total number of objects in the frame at the time the event occurred • new_id<> is the identifier of the detected object
ARCH_DELETE	Record deletion	Deletion of the record from the camera archive from the **Video surveillance monitor**
OPEN_FILE	File opening	Opening a file for playback by a virtual video capture device. Indicates that playback of the next file from the selected folder starts. Among others, the event contains the following parameters: • name<> is the name of the file being played back • tss<> is the time in UTC format in milliseconds from 1/1/1970
CLOSE_FILE	File closing	Finishing a file playback in a virtual video capture device. Indicates that playback of a file is ended. Among others, the event contains the following parameters: • name<> is the name of the file that finished being played back • tss<> is the time in UTC format in milliseconds from 1/1/1970
ARCH_PROTECTED	File protected from rewriting	The event is displayed when a user protects the file from being overwritten. The param<0> parameter contains additional information about the computer name, fragment, user name and user id, which is displayed in the **Add. Info** column in the **Event Viewer** in the following format: "ComputerName;ProtectionPeriod;UserName;UserID", for example, param0<LOCALHOST;04-06-18 16:50:55.612_04-06-18 16:55:55.612;Smith;1>
ARCH_UNPROTECTED	Rewrite protection removed from file	The event is displayed when a user removes protection of the file from being overwritten. The param<0> parameter contains additional information about the computer name, fragment, user name and user id, which is displayed in the **Add. Info** column in the **Event Viewer** in the following format: "ComputerName;ProtectionPeriod;UserName;UserID", for example, param0<LOCALHOST;04-06-18 16:50:55.612_04-06-18 16:55:55.612;Smith;1>

FRAME_SKIPPED	Frames skipped	The FRAME_SKIPPED event occurs in *Axxon PSIM* if there are frames skipped when recording to the archive. By default, the event is generated if there were more than 50 frame skips during the testing period. When frame skipping stops, the FRAME_SKIPPED_STOP event is generated. The description of this event contains information about the number of skipped frames and the time period in which there were skips. By default, both events are generated no more than once every 30 seconds.
FRAME_SKIPPED_STOP	End of frames skipping	The events are controlled by the following registry keys (see Registry keys reference guide): • the FRAME_SKIPPED event can be disabled using the FileSystem.NotifyCoreFrameSkipped registry key; • the delay time between the state changes in seconds is specified in the FileSystem. RecordingStateChangeDelay key. The minimum value is 30 seconds. If you specify a smaller value, the minimum value of 30 seconds will still be used (the key in the registry will be overwritten); • the number of skipped frames for the period, during which the FRAME_SKIPPED event will be generated, is specified in the FileSystem.MaxSkippedFramesByPeriod key
ARCH_BOOKMARKED	Bookmark created	The event is displayed when a user adds a bookmark. The param<0> (i.e. the **Add. Info** column in the **Event Viewer**) parameter contains the comment to the bookmark
ARCH_UNBOOKMARKED	Bookmark deleted	The event is displayed when a user deletes a bookmark
TEMPERATURE_ALARM	Temperature threshold	The param0<> parameter contains the temperature value received from the thermal camera
IGNORE_KEEP_NO_LESS	Ignoring "Keep no less than"	The event is generated when a video fragment is deleted from the archive before expiration of the time period set by the **Keep no less than** parameter. See also Configuring video camera archive depth
AVAILABILITY	Availability of the key position in the license file and the number of objects	The event occurs in response to the GET_AVAILABILITY command and contains information about the number of objects added to the license key for the given position. Parameters: • qty_localpriority<> is the number of allowed objects of the specified type on this computer (local key) • qty<> is the total number of allowed objects of the specified type in the license (generic key) • pos<> is the position number in the license key If there is no position in the key, the event will have a zero value
WRITING_FAILED	Writing error	
FILESYSTEM_FAILED	File system error	Contains mandatory parameter error_msg<>—error text; can also contain parameters cam<>—camera ID and error_code<>—error code. Example: CAM|-1|FILESYSTEM_FAILED|error_msg<Failed to delete index file: D:\\VIDEO\\INDEX\\14061608.idx, error code: 5.>,error_code<5>

The list of commands and parameters for the **CAM** object is given in the following table:

Command—command description	**Parameters**	**Description of parameters**
SETUP—sets (changes) the parameters of a camera	rec_priority<>	Record priority (from 0 to 3, 0—standard, 3—all resources)
	compression<>	Compression ration (0—no compression, 1—maximum quality, …, 5—minimum quality)
	sat_u<>	Value of color (0—min, 10—max)
	proc_time<>	Append period (0–30 sec)
	manual<>	Control brightness and contrast settings (0—manual; 1—auto; 2—auto, but close to values specified manually)
	contrast<>	Contrast (0—min, 10—max)
	md_size<>	Size of motion detection objects (1–16)
	md_mode<>	Mode of pause record (1—enabled, 0—disabled)
	audio_type<>	Type of sound accompaniment
	pre_rec_time<>	Time of pre-record (0–20 sec)
	bright<>	Brightness (0—min, 10—max)
	audio_id<>	Number of a microphone (empty parameter if there is no microphone)
	rec_time<>	Record speed (1–30 FPS, 0—not used)
	alarm_rec<>	Record of alarms (1—enabled, 0—disabled)
	hot_rec_time<>	Time of hot record (0–30 sec)
	hot_rec_period<>	Period of hot record (0–20 sec)
	mux<>	Number of channel (0–1 channel, 15–16 channel)
	color<>	Color (0—black and white, 1—multicolor)
	activity<>	-
	arch_days<>	Number of archive days

	blinding<>	Camera is sealed
	config_id<>	-
	decoder<>	-
	flags<>	Flags
	fps<>	Speed of record (0—not used, 1–30 FPS)
	ifreq<>	Frequency of key frames in sequence (1—each frame is key, 2—100th frame)
	mask0, mask1, mask2, mask3, mask4	Detection tool mask
	md_contrast<>	Sensitivity of motion detection tool (0–15)
	motion<>	Estimation of compressor motion (5–255)
	name<>	Object name
	password_crc<>	Video archive password
	priority<>	Priority of record resource (0—auto, 1—manual)
	resolution<>	Resolution (0—standard CIF, 1—high 2CIF, 2—maximum 4CIF)
	type<>	Type of object
	yuv<>	Color schema of video signal coding (0—YUV4:2:0, 1—YUV4:2:2)
DELETE—disables camera	-	-
START_VIDEO—enables video stream for the current camera	slave_id<>	Name of the computer to which the camera is connected
	comress<>	Level of compression
	register_only<>	-
STOP_VIDEO—disables video stream for the current camera	slave_id<>	Name of computer to which camera is connected
REQUEST_MASK	mask<>	Mask
MUX1, MUX2, MUX3—display the image of a camera on 1, 2, 3 analog outputs	-	-
ACTIVATE—display camera on monitor	monitor<>	Number of a monitor

ARM—arm camera	-	-
DISARM—disarm camera	-	-
REC—start recording from a camera	time<>	Time of record in seconds, if null than only one frame is recorded
	rollback<>	If 1, the record is performed with a rollback
	priority<>	Sets priority of command to start recording. See Appendix 1. Priorities of the start and stop recording commands
	stream_id<>	Sets an identification number of a stream for recording. The stream ID is set as "n.m" where n is the camera ID, m is the number of the stream. *Note. If the specified stream isn't used for any purpose other than record by command (and custom on clients), make sure that the **Lock disabling streams not in use** checkbox is set for it—see The Settings panel of the Camera object*
REC_STOP—stop recording from a camera	priority<>	Sets priority of a command to stop recording. See Appendix 1. Priorities of the start and stop recording commands
	user_id<>	If recording was stopped by a user from the Video surveillance monitor, the parameter contains the user ID. Otherwise the parameter is absent
	from_macro<>	If recording was stopped by a macro, the parameter contains the macro ID. Otherwise the parameter is absent
SET_MASK—set mask	mask<>	Mask
ADD_SUBTITLES—add titles	command<>	Test of imposed titles
	title_id<>	The ID of the **Captioner** object which is used to impose
	page<>	Required parameter to allow recording titles to the titles database to provide search by titles. Available values: BEGIN (start of recording in the database), END (end of recording in the database)
SIP_CONNECT—Sip connected	-	-
SIP_DISCONNECT—Sip disconnected	-	-
SET_IPINT_PARAM—set (change) the parameters of an IP device. Reaction allows changing the IP device settings without going into its web-interface. *Note. For reaction operation it is required to enable the mode of multi-flow video signal—see Configuration of multistream video and Appendix 2. Defining the param_id and param_value values for the SET_IPINT_PARAM reaction*	param_id<>	Name of a parameter. Set of parameters for each camera is individual—see Appendix 2. Defining the param_id and param_value values for the SET_IPINT_PARAM reaction
	param_value<>	Value of a parameter. Set of parameters for each camera is individual—see Appendix 2. Defining the param_id and param_value values for the SET_IPINT_PARAM reaction
	vstream_id<>	Number of a video stream (optional parameter). It is given by “Number of camera”.”Number of a stream”, for example 1.1, 1.2
GET_FRAME—get frame from a camera even if it is not displayed in the Video surveillance monitor	path<>	Path to save a frame. If there is no parameter, the FRAME_SENT event with the data parameter will be formed in the system. Processing of this event is described in The SaveToFile method of the The Script object. Programming using the JScript language

	strea m<>	Optional parameter. Sets *Axxon PSIM* stream to get a frame from. The stream can be specified by a number or purpose. Possible purposes: • stream_archive—stream for archive recording • stream_alarm—stream for archive recording by alarms • stream_client—stream for displaying • stream_analytic—stream for video analytics Stream number consists of a camera ID and a stream ID divided by dot, for example, 4.3 is for stream 3 from camera 4
	time<>	Optional parameter. It is set to request video frame from the archive. Format: DD-MM-YYYY hh:mm:ss. Example: time<19-09-2017 11:35:34>
	gate<>	Optional parameter. Specifies the network name of the **Videogate** from the archive of which to get the frame
	arch<>	Optional parameter. Specifies the network name of the **Backup archive** to get the frame from
	slave _id<>	Optional parameter. Specifies the network name of the Server to get the frame from
	passw ord_c rc<>	Optional parameter. Specifies the CRC sequence to be recorded to the file together with the frame
ARCH_DEL_RECORD—delete archive recordings over the specified period.	fromT ime<>	Mandatory parameter. Time in the YYYY-MM-DDTHH: MM: SS. NNN format, where NNN—milliseconds. The recordings will be deleted (starting with the first one containing the specified time and ending with the last one containing the toTime time). If no time is specified in the toTime parameter, then only one recording will be deleted
	toTim e<>	Optional parameter. Time in the YYYY-MM-DDTHH: MM: SS.NNN format, where NNN—milliseconds. See description above
REC_RESTART—restart recording	-	-
ARCH_BOOKMARK_RECORD—create a bookmark	time1 <>	The date of the archive period beginning included in the bookmark in the DD-MM-YY HH: MM: SS.NNN format, where NNN - milliseconds
	time2 <>	The date of the archive period ending included in the bookmark in the DD-MM-YY HH: MM: SS.NNN format, where NNN—milliseconds
	comm ent<>	Comment to a bookmark
	slave _id<>	Computer and **Video surveillance monitor** IDs using which the bookmark will be created. Parameter format: <computer id>.<monitor id>. For example, slave_id<WS2.1>—WS2 is a computer ID and 1 is Video surveillance monitor ID
CRUISE_START—auto cruise	cruise _id<>	Route name on a camera
	action <>	Executed action: • CRUISE_START—start cruising along the specified route • PATROL_PLAY—start patrolling along the specified route
	cam_i d<>	Camera ID

GET_DEPTH—get the archive depth. The ARCHIVE_DEPTH event from the SLAVE object (see SLAVE Computer) is created in the system as the response to this reaction. If one or both parameters are absent it means that there is the archive depth request for all possible parameters	drive<>	Disk or network path to request the archive depth. The disk name is set in the "<disk letter>:\\" format, for example drive<D:\\> *Note. The "\" character is an escape character.* The network path is set in the UNC format
	arch	Get the backup archive depth. Example. `DoReactStr("CAM","2","GET_DEPTH","drive<D:\\>,cam<2>,arch");`
	gate	Get the videogate archive depth. Example. `DoReactStr("CAM","1","GET_DEPTH","drive<V:\\>,gate");`
CLEAR_SUBTITLES—remove all titles from the video image	title_id<>	The ID of the **Captioner** object
GET_AVAILABILITY—check the availability of the key position	pos<>	The position number in the license key

Properties of the **CAM** object are shown in the table.

Properties of the CAM object	**Description of the object properties**
ID<>	Object ID
PARENT_ID<>	Parent object ID
TELEMETRY_ID<>	Telemetry module ID (PTZ ID)
REGION_ID<>	Region ID

The **CAM** object can be in the following states.

State of CAM object	**Description**
ALARMED	Camera is in alarm mode.
DISARM_DETACHED	No signal from camera
DETACHED	No signal from camera
ARMED	Camera is armed
DISARMED	Camera is disarmed

MONITOR Monitor

The **MONITOR** object corresponds to the **Monitor** system object.

The **MONITOR** object sends events presented in the table. Procedure is started when the corresponding event occurs.

List of commands and parameters for the **MONITOR** object is presented in the following table:

Event	Description	Comment
STARTED_AVI_EXPORT	Video export started	Among others, the event has the following parameters: • slave_id<>—operator who started the export • param1<>—number of camera on which the export is performed, date and time of export period beginning. The parameter value is like "<RecNo.> Camera <id> (dd-mm-yy hh:mm:ss)", for example param1<01 Camera 1 (05-10-17 10:23:21)> • time<>—time when export started
FINISHED_AVI_EXPORT	Video export finished	Among others, the event has the following parameters: • slave_id<>—operator who started the export • param1<>—number of camera on which the export is performed, date and time of export period ending. The parameter value is like "<RecNo.> Camera <id> (dd-mm-yy hh:mm:ss)", for example param1<01 Camera 1 (05-10-17 10:23:21)> • time<>—time when export ended • param<0>—additional information displayed in the corresponding column of the Event Viewer, in the following format: "ComputerName;ExportPeriod;UserName;UserID", for example, param0<LOCALHOST;04-06-18 16:50:55_04-06-18 16:55:55;Smith;1>
AVI_EXPORT_RESULT	Video export result	The event has the same parameters as START_AVI_EXPORT with additional error_result<> having one of the following values: 0—export successful 1—unknown 2—busy 3—not ready 4—invalid interval 5—file error
PLAY_START	Start the archive fragment playback	-
PLAY_STOP	Stop the archive fragment playback	-
INTERFACE_MANIPULATION	Visualization change	param<0>—additional information displayed in the corresponding column of the Event Viewer, contains the identifier of the camera that was moved around the layout
LAYOUT_DEL	Deleting layout	param<0>—additional information displayed in the corresponding column of the Event Viewer, contains the name of the deleted layout
LAYOUT_ADD	Adding layout	param<0>—additional information displayed in the corresponding column of the Event Viewer, contains the name of the added layout
LAYOUT_ACTIVATE	Changing active layout	param<0>—additional information displayed in the corresponding column of the Event Viewer, contains the name of the activated layout
REPLACE_CAM	Changing the camera position	param<0>—additional information displayed in the corresponding column of the Event Viewer, in the following format: <Camera 1 name> <Camera 2 name>
ACTIVATE_CAM	Camera activated	auto_switch<>—indicates whether slide show (auto paging, auto scrolling) was enabled at the time of camera activation. This parameter can be used to turn off slide show when activating a Video surveillance window

CAM_EXPAND	The Video Surveillance Window expanded to the entire Monitor	The event is generated if the following registry keys are set (see Registry keys reference guide): • MaximizeCameraOnDblClk=1 • MinimizeCameraOnDblClk=1 Event parameters: • param0<>—camera ID • user_id<>—the ID of the user who performed the action
CAM_COLLAPSE	The Video Surveillance Window collapsed back	The event is generated if the following registry keys are set (see Registry keys reference guide): • MaximizeCameraOnDblClk=1 • MinimizeCameraOnDblClk=1 Event parameters: • param0<>—camera ID • user_id<>—the ID of the user who performed the action

List of commands and parameters for the **MONITOR** object is presented in the table:

Command—command description	**Parameters**	**Description**
"REMOVE"—removes camera from monitor	cam<>	ID of camera in the settings tree which must be removed from monitor
	show<>	Optional parameter. Possible values: • 0—do not update the layout in the Monitor after removing the camera. There may be empty space not occupied by Video surveillance windows • 1—update the layout in the Monitor after removing the camera to minimize empty space
"REMOVE_ALL"—removes all cameras from monitor	-	-
"STOP_VIDEO"—stops video stream of camera	cam<>	ID of a camera in the settings tree, the video stream from which must be stopped
"REPLACE"—removes all cameras from monitor and triggers the specified camera	slave_id<>	Name of a computer to which monitor belongs, it is possible to place owner in script
	cam<>	ID of a camera in the settings tree which must be displayed in the monitor
	name<>	Name of a camera which will be displayed in the bottom-left corner
	audio_type<>	-
	audio_id<>	-
	arch_id<>	-
	control<>	0—only archive viewing, 1—it is also possible to control (arm/disarm, record)
"ADD_SHOW"—adds cameras on the monitor	cam<>	ID of a camera in the settings tree which must be displayed in the monitor

Note. See also PLACE_CAM_IN_LAYOUT_CELL	name<>	Object name which will be displayed in the bottom-left corner
	arch_id<>	-
	control<>	0—only archive viewing, 1—it is also possible to control (arm/disarm, record)
	gate_id<>	ID of the videogate through which you want to receive video. The corresponding camera must be added and configured in this videogate—see Selecting and configuring the cameras for the Videogate module
	slave_id<>	ID of a computer to which the command is applied
"ACTIVATE_CAM"—activates camera	cam<>	ID of a camera in the settings tree which must be activated
"ARCH_FRAME_TIME"—search for video archive by date and time	cam<>	-
	date<>	-
	time<>	-
	mode<>	Can take the following values: • 0—videogate, if it is set (if not set, then the archive of the video server is searched) • 1—sideo server • 2—backup archive • 10 + Object ID External storage in the Monitor object settings panel (normally 11)—external storage
"SETUP"—sets parameters of monitor	no_update<>	-
	overlay<>	Disable the mode of speed-up displaying
	x<>	Coordinate of top-left corner (0–100)
	y<>	Coordinate of top-left corner (0–100)
	w<>	Size in horizontal direction (0–100)
	h<>	Size in vertical direction (0–100)
	max_cams<>	Maximum allowable number of cameras on the monitor
	min_cams<>	Minimum allowable number of cameras on the monitor
	compress<>	-
	panel<>	Show control panel (0—disabled, 1—enabled)

	panel_type<>	-
	s<>	-
	layout<>	-
	gate<>	-
	map_id<>	-
	enable<>	-
	topmost<>	1—show screen always on top
	type<>	Type of **Monitor** object
	allow_move<>	Allows moving of window
	arch_id<>	Archive ID
	cycle<>	Delay when auto scrolling (1–20 sec)
	flags<>	Flags
	name<>	Name of object
	overlay<>	Enable the mode of speed-up displaying. (0—no speeding-up, 1—"overlay mode" speeding-up, 2—"DirectDraw mode" speeding-up)
	tel_prior<>	Telemetry priority
	gstream_version<>	If the value is not set, the function for stream auto select is disabled If the value is **minBPS**, then the stream for displaying is selected automatically as described in Configuring automatic selection of video stream for display
"ACTIVATE"—activates control panel of monitor	user_id<>	User ID
	panel_active<>	-
"DEACTIVATE"—deactivates control panel of monitor	-	-
"EXPORT_FRAME"—exports frame in a JPG file	cam<>	-
	file	-

"KEY_PRESSED"—controls buttons of video surveillance monitor and video records archive	number <>	-
	cam_id <>	The ID of the camera to the Video surveillance window of which the command must be applied. If the identifier is not specified, the command is applied to the active Video surveillance window (see Active Surveillance window)
	key <>	Possible values: "ARCH_EDIT_DATE"—change date of search by archive; "ARCH_EDIT_TIME"—change time of search by archive; "ARCH_EDIT_ENTER"—enter changes of values in archive; "ARCH_EDIT_ESCAPE"—cancel editing of archive; "ARCH_EDIT_BACK"; "ARCH_EDIT_REPLACE"; "WINDOW_ZOOM_IN"—expand window of video surveillance; "WINDOW_ZOOM_OUT"—hide window of video surveillance; "ZOOM_IN"—image incoming; "ZOOM_OUT"—image removal; "CYCLE_REW"—scrolling video surveillance windows back; "CYCLE_FF"—scrolling video surveillance windows forward; "LEFT"—move the frame left in the Zoom mode; "RIGHT"—move the frame right in the Zoom mode; "UP"—move the frame up in the Zoom mode; "DOWN"—move the frame down in the Zoom mode; "MODE_VIDEO"—video surveillance mode; "MODE_ARCH"—mode of archive video records playback; "MODE_ARCH2"—mode of archive video records playback 2; "MASK_SHOW"—show mask; "MASK_HIDE"—remove mask; "ARM"—arm camera; "DISARM"—disarm camera; "REW"—rewind; "PLAY"—play; "PLAY_NONSTOP"—non-stop playback; "PLAY_FAST"—speed up video record playback; "FF"—fast forward; "RECORD"—record; "RECORD_MIC"—record from microphone; "STOP"—stop; "REC_STOP"—stop record;

"PAUSE"—pause;

"MIC_ON"—microphone On;

"MIC_OFF"—microphone Off;

"PRINT"—print the frame;

"SELECT_LAYOUT"—control layout of video surveillance monitor;

"START_CYCLE_FF"—enable automatic forward scrolling of video surveillance windows. Period of scrolling video images is specified when configuring the **Monitor** interface object (see Configuring the display mode of camera windows);

"STOP_CYCLE"—stop slide show of Video surveillance windows;

"EXPORT_DO"—open the AviExport tool for background export (see The AviExport utility);

"PROTECT_DO"—open the dialog box to create a bookmark (see Create a bookmark);

"PROTECT_VIEW"—show the bookmark list (see List of bookmarks)

"START_AVI_EXPORT"—starts video export *Note. See the examples in Examples with Cameras and Video surveillance monitors*	start <>	Start time
	finish<>	End time
	avi_path <>	Path to created file
	cam <>	Camera ID
"STOP_AVI_EXPORT"—stops video export	monitor <>	Number of monitor
"START_AVI_SCHEDULE"—starts bookmarks export	-	-
"STOP_AVI_SCHEDULE"—stops bookmarks export	-	-
"CONTROL_TELEMETRY"—Telemetry control. See Mouse PTZ control	cam <>	ID of a camera on which you want to enable or disable the mouse PTZ control
	on<>	0—disable mouse PTZ control 1—enable mouse PTZ control
"SET_REC_RESTART"—set recording restart when entering the archive	-	-
"RESET_REC_RESTART"—reset recording restart when entering the archive	-	-
"SET_ARCH_ENTER_PAUSE"—enable playback pause when entering the archive	-	-
"RESET_ARCH_ENTER_PAUSE"—disable playback pause when entering the archive.	-	-
"DISABLE_TELEMETRY"—disable telemetry control from Video surveillance monitor	-	-
"ENABLE_TELEMETRY"—enable telemetry control from Video surveillance monitor	-	-
"INCREASE_VIEW"—increase camera window size in the Video surveillance monitor	cam <>	Camera identifier
"DECREASE_VIEW"—decrease camera window size in the Video surveillance monitor	cam <>	Camera identifier
"SHOW_LAYOUT"—show layout with the specified ID	layout_id<>	Layout ID in database
"GO_LIVE"—switch all cameras on the monitor to live video mode	-	-

"GO_ARCH"—switch all cameras on the monitor to archive mode	arch_time<>	Optional parameter. Sets the time position in the archive in the DD-MM-YY HH:MM:SS format. By default, the archive is positioned on the last record
"SAVE_AS"—export selected archive fragment	-	-
PLACE_CAM_IN_LAYOUT_CELL—add camera to the specific cell on the specific layout on the Monitor	cam<>	ID of a camera in the objects tree which must be displayed in the monitor. If the parameter value is incorrect, for example, 0 or -1, then the corresponding cell will be hidden
	layout_name<>	ID or name of the layout to add camera on
	cell<>	The number of the cell on the layout to which the camera must be added. Cells are numbered from top to bottom left to right, starting from the top-left corner of the layout. ***Attention!*** *Cells are numbered starting from 0.* If some other camera has already been added to the specified cell, it will be replaced
SET_TITLES—shows captions over a video image in any display mode. Such captions are not archived and are displayed until CLEAR_TITLES command is applied or Monitor is reset	cam<>	The ID of a camera to the Video surveillance window of which the command must be applied
	titles<>	The caption text that must be displayed. Use '\r' to break the line
	title_id<>	Captioner ID
CLEAR_TITLES—disable the captions created with the help of the SET_TITLES command	cam<>	The ID of a camera to the Video surveillance window of which the command must be applied
	title_id<>	Captioner ID

Properties of the **MONITOR** object are displayed in the table.

Properties of the MONITOR object	**Description of properties**
ID<>	Object ID
PARENT_ID<>	Parent object ID

MACRO Macro

The **MACRO** object corresponds to the **Macro** system object.

The **MACRO** object sends events presented in the table. Procedure is started when the corresponding event occurs.

Event	Description	Parameters	Parameter description
RUN	Action is performed	src_sender<>	The name of the computer on which the macro was executed. *Note. The value of this parameter is displayed in the **Add. info** column of the Event Viewer in real time. When Axxon PSIM is restarted and event log records are loaded from the database, this information is not displayed in the interface, but remains in the database*
		user_id<>	The identifier of the user who executed the macro. *Note. The value of this parameter together with the username is displayed in the **Add. info** column of the Event Viewer in real time. When Axxon PSIM is restarted and event log records are loaded from the database, this information is not displayed in the interface, but remains in the database*

List of commands and parameters for the **MACRO** object is presented in the following table:

Command—command description	Parameters	Description
"RUN"—performs an action	-	-
"SETUP"—sets parameters for a macro	name<>	Object name
	flags<>	Flags
	state<>	Object state
	hidden<>	«Hidden» flag
	local<>	«Local» flag

Properties of the **MACRO** object are shown in the table.

Properties of the MACRO object	Description of properties
ID<>	Object ID
PARENT_ID<>	Parent object ID

The **MACRO** object can be in the following states:

State of the MACRO object	Description
"NORM"	Normal

SLAVE Computer

The **SLAVE** object corresponds to the **Computer** system object.

The **SLAVE** object sends events presented in the table. Procedure is started when the corresponding event occurs.

Events	Description	Comment
CONNECTED	Connecting	Event is generated when a Client is connected to the Server
DISCONNECTED	Disconnecting	Event is generated when a Client is disconnected from the Server
KEY_IGNORED_HW	Key ignored (mismatch of card codes)	Event is generated if codes of cards (or HID) in the key mismatch to current codes of computer
KEY_IGNORED_SW	Key ignored (limitation exceeded)	Event is generated if there software limitations. For example, if key is accepted but number of created objects in the objects tree is more than specified in the key
KEY_UPDATED	Key updated	
PROTOCOL_RCVD	Protocol received	
REBUILD_IN_START	Start of archive re-indexing	
REBUILD_IN_STOP	End of archive re-indexing	
REGISTER_ATTEMPT	Attempt of unauthorized access	
REGISTER_ERROR	Limit of access attempts is exceeded	Event is generated when user failed to enter the system a lot of times. Some timeout is started after the event when user can’t try to enter the system. Number of attempts and timeout can be changed using registry
REGISTER_USER	User registration	This event is generated when user tries to enter the system (when entering login and password)
DISC_EXIST	Disk for archive record exists	
NO_DISC	There is no disk for archive record	
KEY_IGNORED_FR	Key ignored	Event is generated in case of key file is not recorded on the disc
SHUTDOWN	Shutdown	

DISC_MOUNT	Disc connected (mounted)	
DISC_UNMOUNT	Disc disconnected (unmounted)	
ARCHIVE_DEPTH	Archive depth	Event is generated at midnight and contains information about archive depth on all disks in hours (the depth<> parameter). To call the event manually, use the GET_DEPTH reaction. Archive depth in the Days:Hours format is specified in the **Additional information** field of the Event Viewer when displaying an event. Also this information contains in parameter of the param0<> event. Archive depth is counted as discrepancy between date of creation the oldest archive file and date of creation the newest archive file (on disk or by camera)
FORCED_OFF	Forced offload	The event is generated before the forced unload of *Axxon PSIM*, for example, if the Guardant security key is removed. Unload is performed after the action caused it (for example, removing the Guardant key) after the time specified by the UnloadDelay registry key—see Registry keys reference guide
DEACTIVATE_ALL_DISP	Hide all displays	The event is generated when **Hide all** command is executed on a computer set in slave<> parameter. If except<> parameter is present, all displays except the one specified in this parameter are hidden
LIC_EXPIRATION	License expires in	Not generated by default. Set NotifyExpireLic = 1 to enable (see Registry keys reference guide). The days<> parameter shows the number of days left till license expiration (can be non-integer). The event is generated at *Axxon PSIM* start-up and when the day changes
DATABASE_ERROR	Database connection lost	The event is generated when the connection to SQL Server is lost the first time it is accessed after the disconnection
SCRIPT_ERROR	Script execution failed	By default, the event is not generated, since it is not added to the psim.ddi external settings file. In order for the SCRIPT_ERROR event to be generated and added to the PROTOCOL table, it must be added to the psim.ddi table (see Editing the external setting file (Axxon PSIM.ddi) using the ddi.exe utility)

List of commands and parameters for the **SLAVE** object is presented in the following table:

Command—command description	**Parameters**	**Description**
"**SETUP**"—set parameters for a computer	display_id<>	Display ID
	drives<>	Disks for record of video archive
	drives_a<>	Disks for record of audio information
	flags<>	Flags
	arch_days<>	Size of event archive

	connection<>	Connection
	disable_protocol<>	Disable protocol
	ip_address<>	IP-address of device
	is_backup<>	Backup
	is_load<>	Loaded
	local_protocol<>	Local protocol
	modem<>	Modem connection
	name<>	Object name
	password<>	Password
	sync_time<>	Time synchronization
	username<>	User name
"BACKUP"—backup database	-	-
"CONNECT_ONE"—connect to a computer. Connects the corresponding computer. It is recommended to avoid using of this reaction manually	-	-
"CONNECT_OTHER"—connect to cores. Connects computer to other cores from configuration. It is recommended to avoid using of this reaction manually	-	-
"DISCONNECT_ONE"—disconnect from computer. Disconnects the corresponding computer. Core can be connected automatically in case of disconnection. It is recommended to avoid using of this reaction manually	-	-
"SYNC_PROTOCOL"—run SyncProtocol.exe utility of protocol synchronization. Protocol merging is happened if synchronization is configured	-	-

"SYNC_TIME"—synchronize time. To perform this reaction it is required to create SyncTime parameter with value 1 on the system to which reaction was addressed in the HKEY_LOCAL_MACHINE\SOFTWARE\AxxonSoft\PSIM (HKEY_LOCAL_MACHINE\Software\Wow6432Node\AxxonSoft\PSIM registry section for 64-bits system)	-	-
"CREATE_PROCESS"—run process	command_line<>	Command line. Commands of Windows command line written without hyphens through |, & or && separating characters
"SEND_MY_CONFIG"—send configuration. Send configuration to other computers. The same as "SPREAD_CONFIG"	-	-
"MOVE_CONFIG"—move configuration. Moves configuration created in the objects tree on the basis of the computer–supplier to the computer–recipient	from<>	Supplier
	to<>	Recipient
"SPREAD_CONFIG"—spread configuration. The same as "SEND_MY_CONFIG"	-	-
"GET_DEPTH"—get archive depth. The ARCHIVE_DEPTH event (see table below) is formed in response to reaction in system. Absence of one or both parameters concedes request of depth by records for all values of parameter	cam<>	Camera ID for which archive depth is required
	drive<>	Disk or network path on which archive depth is required. Name of disk is specified in the following format: "<letter of disk>:\\", for example drive<D:\\> *Note. The "\" symbol is an escape character.* Network path is specified in the UNC format
"ACTIVATE_DISPLAY"—change the display. The command allows showing the Display with the given identifier on the monitor (monitors) of the computer	display_id<>	The identifier of the corresponding **Display** object. If an empty value is passed to the parameter, then all displays are hidden when running this command

Properties of the **SLAVE** object are shown in the table.

Properties of the SLAVE object	**Description**
ID<>	Object ID
PARENT_ID<>	Parent object ID
USER_ID<>	User ID

DISPLAY Display

The **DISPLAY** object corresponds to the **Display** system object.

The events presented in the table come from the **DISPLAY** object. Procedures are run when the corresponding event occurs.

Event	Description
ACTIVATE	Display is activated
DEACTIVATE	Display is deactivated
ACTIVATED	Display is activated on the remote computer. The computer name is transferred in the param0 <> parameter

List of commands and parameters for the **DISAPLY** object is presented in the following table:

Command—command description	Parameters	Description
ACTIVATE—show display	macro_slave_id<>	Name of a computer on which display must be shown
DEACTIVATE—hide display	macro_slave_id<>	Name of a computer on which display must be hidden

Note

If the "macro_slave_id" parameter is not set, the command will be performed for all computers in the system.

Properties of the **DISPLAY** object are shown in the table.

Properties of the DISPLAY object	Description
flags	Flags
id	Object ID
name	Object name
parent_id	Parent object ID

PLAYER Audio player

The **PLAYER** object corresponds to the **Audio player** system object.

List of commands and parameters for the **PLAYER** object is presented in the table.

Command—command description	**Parameters**	**Description**
"PLAY_WAV"—plays back the audio file	file<>	Full path to the audio file in the .wav format (with the name of the file being played back. For example: C:\Program Files (x86)\Axxon PSIM\Wav\cam_alarm_1.wav)
	from_macro<ID>	Response flag of the audio file being played back. The response is sent from the macro (specifying the ID of the existing macro, ID > 0). *Note. The parameter isn't mandatory if the voice notification channel is configured in the **Audio player** object interface (see Setting up the voice notification using Audio player object)*
"SETUP"—sets the audio player parameters	board<>	Sound unit of the archive player
	flags<>	Flags
	h<>	Height of settings dialog (0–100)
	name<>	Object name
	voice<>	Sound notification
	voice_board<>	Sound unit of notification
	w<>	Width of settings dialog (0–100)
	x<>	Left top corner of settings dialog (0–100)
	y<>	Left top corner of settings dialog (0–100)
"STOP_WAV"—stops audio file playback	-	-

Properties of the **PLAYER** object are given in the following table.

Properties of the PLAYER object	**Description of properties**
ID<>	Object ID
PARENT_ID<>	Parent object ID

CORE

The **CORE** object is the core of the system, a global static object that implements methods used to control the state and manage system objects of *Axxon PSIM*. The extended possibilities for working with the CORE object are given when using scripts in the JScript programming language—see The Script object. Programming using the JScript language.

The **CORE** object sends events presented in the table.

Event	**Description**
DO_REACT	Event triggers the reaction of some object in the system. The action parameter of this event contains a description of the action that must be performed. Examples of values of the action parameter: SET_MARKRECT—sent when a face is recognized on the video image; DEL_MARKRECT—sent when a face disappears from the video image. Other event parameters may also be present that can be monitored using the Debug window (see The Debug window). If the value of the is SET_MARKRECT, then the param5_val parameter contains the number of the camera on which the face was detected in the video image. This is indicated by the name of the parameter forwarded in the param5_name parameter. For the DEL_MARKRECT value, the number of the camera is forwarding in the param0_val parameter
SLAVE_CHANGED	Event is generated when Failover service becomes active. It consists of the following parameters: • old_slave_id—ID of the **Computer** object from which cameras are transferred • new_slave_id—ID of the **Computer** object to which cameras are are transferred • CAM<n1, n2, ... >—where n1, n2, and so on are IDs of cameras transferred to another **Computer** parent object. For example, CAM<4,6,7>—transfer cameras with IDs 4, 6, 7
CREATE_OBJECT	Event triggers creation of an object. Parameters: • objtype<>—object type, for example, objtype<PERSON> for user creation • parent_id<>—identification number of the parent object • service_photo<>—when a user is created, the base64-encoded binary image of user photo can be put to this parameter. This is necessary for adding user photo immediately whet creating user in Access Manager

MAP Map

The **MAP** object corresponds to the **Map** system object.

The **MAP** object sends the events presented in the table. The procedure is started when the corresponding event occurs.

Event	Description
LAYER_ACTIVATED	Layer activation. This event is received when a layer is selected on the Map. The obj_id<> parameter has the ID of the activated layer
ACTIVATE_OBJECT	Object activation. The event is received when an object is selected (activated by mouse click) on the Map. Parameters: 1. obj_type <>—object type 2. user_id<>—user ID 3. module<>—module name, for the Map—map.run 4. date<>—date when the event occurred 5. time<>—time when the event occurred 6. slave_id<>—computer network name 7. obj_id <>—object ID 8. layer<>—Map layer ID 9. fraction<>—millisecond when the event occurred 10. owner<>—user who activated the object 11. type_of_display <>—object display type, possible values: a. IMAGE—image b. IMAGE_AND_INDICATOR—image and indicator c. TEXT—text d. LINE—line e. POLYGON—polygon f. ELIPSIS—ellipse g. TITLE—the name of the object
OBJDBLCLK	The event is received when you double click an object on the Map. Contains the same parameters as ACTIVATE_OBJECT

The list of commands and parameters for the **MAP** object is presented in the table.

Command	Parameters	Description
SET_TOPMOST—Set topmost	-	-
SET_NOTOPMOST—Cancel topmost	-	-
HIDE_OBJECT—Hide/show object icon on the map	objtype<>	Object type. Can be left blank. If the object type is not set, then objects of all types are hidden/shown
	objid <>	Object ID. Can be left blank. If the object ID is not set, then all objects of the specified type are hidden/shown
	hide <>	0—objects are shown on the map 1—objects are hidden on the map
SET_OBJECT_GEOMETRY—Set object location on the map	objtype<>	Object type
	objid <>	Object ID
	x<>	New coordinate of the top left corner of the object icon on the map layer along the X axis in pixels
	y<>	New coordinate of the top left corner of the object icon on the map layer along the Y axis in pixels

	exclude_children<>	By default, when using the SET_OBJECT_GEOMETRY reaction, when moving the object icons, the names of these objects (child objects) also move. If you pass the exclude_children <1> parameter in the reaction, then the object is moved separately from the children, that is, without their names
INSCRIBE—Inscribe to window	-	-
SHOW_MINIMAP—Show a minimap	x<>	The coordinate of the top left corner of the minimap along the X axis in pixels
	y<>	The coordinate of the top left corner of the minimap along the Y axis in pixels
	w<>	Width of the minimap in pixels
	h<>	Height of the minimap in pixels
	monitor<>	Monitor ID
	slave_id<>	Computer network name
SET_ZOOM—Change the Map scale	zoom<>	Map scale ratio
ACTIVATE_OBJECT—Activate object on the Map	obj_type<>	Object type
	obj_id<>	Object ID
	layer<>	Map layer ID. If the parameter is set, the script will work on the specified layer. If the parameter is not set, the scrip will work on the current layer
DRAW_ARROW—Draw a track of movement between objects	first_obj_type<>	Type of the object from which a track will be made
	first_obj_id<>	ID of the object from which a track will be made
	second_obj_type<>	Type of the object to which a track will be made
	second_obj_id<>	ID of the object to which a track will be made
	obj_id<>	ID of the created track
	title_text<>	Text that will be displayed next to the track. \n is used for line break. Additional optional parameters: • title_text_align—shift of the text. 0—text is displayed in the center, 1—at the beginning of the track, 2—at the end of the track • title_text_size—size of the text • title_shift_by_y—shift down of the text (may be needed when using title_text_size). Default is 40 pixels • title_text_color—color of the text in Decimal format • title_text_font—font of the text Example: title_text<Object is moving\nto the exit>,title_text_align<1>,title_shift_by_y<60>, title_text_color<16711935>,title_text_size<18>,title_text_font<Algerian>

ERASE_ARROW— Erase a track of movement between objects	obj_id <>	ID of the track that must be erased. If you don't specify the parameter, all tracks will be erased

Features of the **DRAW_ARROW** command execution:

1. When executing the command, a track will be displayed on each layer of the **Map** as arrows  between the objects.
2. If the objects are located on the same layer, the arrow is drawn directly between the specified objects. If the objects are located on different layers, the arrow is drawn by the shortest path.
3. You can limit the depth of searching for connections between layers to make a track with the DrowArrowSearchDepth key, see Registry keys reference guide.
4. If
    a. it isn't possible to make a track,
    b. one of the objects doesn't exist,
    c. it is possible to make a track, but the arrows cannot be displayed,

then the icon  will be displayed on the first object, and the icon  will be displayed on the second object.

OLXA_LINE Microphone

The **OLXA_LINE** object corresponds to the **Microphone** system object.

The **OLXA_LINE** object sends events presented in the table. Procedure is started when the corresponding event occurs.

Event	Description
ACCU_START	Sound activated recording is on
ACCU_STOP	Sound activated recording is off
ARM	Recording is on
DISARM	Recording is off
INCOMING_NUMBER	Incoming telephone number
OUTGOING_NUMBER	Outgoing telephone number
REC	Start of recording
REC_STOP	End of recording
RESET	Microphone connecting

List of commands and parameters for the **OLXA_LINE** object is presented in the following table:

Command—command description	Parameters	Description
"ARM"—microphone is recording	-	-
"DISARM"—microphone isn't recording	-	-
"SETUP"—sets microphone parameters	type<>	Type of line
	accu_start <>	Sound detection threshold
	accu_stop<>	Holding time of detection triggering
	amp<>	Amplification
	aru<>	Automatic amplification control
	aru_dyn<>	Level of AGC
	aru_time<>	AGC attack time
	chan<>	Number of microphone sound channel
	compression<>	Type of compression
	flags<>	Flags
	name<>	Object name
	rec<>	Start of recording

Properties of the **OLXA_LINE** object are given in the table.

Properties of the OLXA_LINE object	Description of properties
ID<>	Object ID
PARENT_ID<>	Parent object ID

The **OLXA_LINE** object can be in the following states:

State of the OLXA_LINE object	State description
"BLUE"	Microphone disarmed

"GREEN"	No signal from microphone
"YELLOW"	Microphone armed
"RED"	Start of recording

TELEMETRY PTZ device

The **TELEMETRY** object corresponds to the **PTZ device** system object.

The **TELEMETRY** object sends events presented in the table. The procedure is started when the corresponding event occurs.

Event	Description	Comment
LOCKED	Locked	Event is received after the LOCK command (see the table below)
UNLOCKED	Unlocked	Event is received after the UNLOCK command (see the table below)

List of commands and parameters for the **TELEMETRY** object is presented in the following table:

Command—command description	Parameters	Description
AUTOFOCUS_ON—enable autofocus	tel_prior <>	Priority (1—low, 2—medium, 3—high)
AUTOPAN_END_P—specify the end point of autopan	tel_prior <>	Priority (1—low, 2—medium, 3—high)
AUTOPAN_START—start autopan	tel_prior <>	Priority (1—low, 2—medium, 3—high)
AUTOPAN_START_P—specify the start point of autopan	tel_prior <>	Priority (1—low, 2—medium, 3—high)
AUTOPAN_STOP—stop autopan	tel_prior <>	Priority (1—low, 2—medium, 3—high)
CLEAR_PRESET—clear the selected preset	tel_prior <>	Priority (1—low, 2—medium, 3—high)
	preset <>	Preset
D2OFF—disable the additional dynamic settings for Panasonic PTZ video cameras used to increase the quality of analog video signal	tel_prior <>	Priority (1—low, 2—medium, 3—high)
D2ON—enable the additional dynamic settings for Panasonic PTZ video cameras used to increase the quality of analog video signal	tel_prior <>	Priority (1—low, 2—medium, 3—high)
DOWN—rotate video camera lens down	tel_prior <>	Priority (1—low, 2—medium, 3—high)

FOCUS_IN—zoom in	tel_prior <>	Priority (1—low, 2—medium, 3—high)
FOCUS_OUT—zoom out	tel_prior <>	Priority (1—low, 2—medium, 3—high)
FOCUS_STOP—stop zooming in/out of image	tel_prior <>	Priority (1—low, 2—medium, 3—high)
GO_PRESET—rotate video camera to the position specified on the preset	tel_prior <>	Priority (1—low, 2—medium, 3—high)
	preset <>	Preset
HOME—rotate video camera to the initial (home) position	tel_prior <>	Priority (1—low, 2—medium, 3—high)
IRIS_CLOSE—close diaphragm	tel_prior <>	Priority (1—low, 2—medium, 3—high)
IRIS_OPEN—open diaphragm	tel_prior <>	Priority (1—low, 2—medium, 3—high)
IRIS_STOP—stop diaphragm	tel_prior <>	Priority (1—low, 2—medium, 3—high)
LEFT—rotate video camera lens to the left	tel_prior <>	Priority (1—low, 2—medium, 3—high)
LEFT_DOWN—rotate video camera lens to the left and down	tel_prior <>	Priority (1—low, 2—medium, 3—high)
LEFT_UP—rotate video camera lens to the left and up	tel_prior <>	Priority (1—low, 2—medium, 3—high)
PATROL_LEARN—start procedure of patrol programming performed by recording the video camera actions	tel_prior <>	Priority (1—low, 2—medium, 3—high)
PATROL_PLAY—start patrolling	tel_prior <>	Priority (1—low, 2—medium, 3—high)

PATROL_STOP—stop patrolling	tel_prior<>	Priority (1—low, 2—medium, 3—high)
RIGHT—rotate video camera lens to the right	tel_prior<>	Priority (1—low, 2—medium, 3—high)
RIGHT_DOWN—rotate video camera lens to the right and down	tel_prior<>	Priority (1—low, 2—medium, 3—high)
RIGHT_UP—rotate video camera lens to the right and up	tel_prior<>	Priority (1—low, 2—medium, 3—high)
SET_PRESET—record the current position of video camera to the selected preset	tel_prior<>	Priority (1—low, 2—medium, 3—high)
	preset<>	Preset
STOP—stop video camera lens rotation	tel_prior<>	Priority (1—low, 2—medium, 3—high)
UP—rotate video camera lens up	tel_prior<>	Priority (1—low, 2—medium, 3—high)
SETUP—set up PTZ device	address<>	Device address
	cam<>	Camera ID to control
	flags<>	Flag of object operating (0—ON, 1—OFF)
	name<>	Object name of PTZ device
	speed<>	Speed
SEND_BUFFER—send command to COM port in the hexadecimal format	buffer<>	Command in the hexadecimal format
	parent_id<>	ID of **Telemetry Controller** parent object. The required parameter
	tel_prior<>	Priority (1—low, 2—medium, 3—high). The value of the parameter must be greater than 0

LOCK—lock. Switch the telemetry over to the LOCKED state for a specified time	tel_prior<>	Priority (1—low, 2—medium, 3—high). The value of the parameter must be greater than 0. It is forbidden to perform control commands with a lower priority than the specified during the lock time
	duration<>	Lock duration. If the parameter is not specified, the lock is valid until the UNLOCK command is executed
UNLOCK—unlock. Switch the telemetry over to the UNLOCKED state for a specified time	-	-
AUTOFOCUS_OFF—disable autofocus	tel_prior<>	Priority (1—low, 2—medium, 3—high).  To use this command, you must add it to the **Reactions** tab for the **TELEMETRY** object in ddi.exe (see The Reactions tab)

Properties of the **TELEMETRY** object are shown in the table.

Properties of the TELEMETRY object	**Description of the object properties**
ID<>	Object ID of the PTZ device
PARENT_ID<>	Parent object ID

The **TELEMETRY** object can be in the following states:

State of the TELEMETRY object	**Description of the object state**
LOCKED—locked	Control of telemetry is locked with some priority. It is forbidden to control telemetry with a priority lower than the specified when locking (see the table above)
UNLOCKED—unlocked	It is allowed to control telemetry with any priority

TELEMETRY_EXT Keyboard

The **TELEMETRY_EXT** object corresponds to the **Keyboard** system object.

The **TELEMETRY_EXT** object sends events presented in the table. Procedure is started when the corresponding event occurs.

Event	Description of event	Parameter	Description of parameter	Value range
KEY_PRESSED	Key is pressed	param 0<>	Code of pressed key	See Installing and configuring security system components guide
		device <>	Device on which key is pressed	0—*AXIS T8312* main keyboard 1—*AXIS T8313* keyboard
KEY_RELEASED	Key is released	param 0<>	Code of released key	0..21 for *AXIS T8312*. For *BOSCH KBD-Digital, BOSCH KBD-Universal* and *Panasonic WV-CU950* see Installing and configuring security system components guide
		device <>	Device on which key is released	0—*AXIS T8312* main keyboard 1—*AXIS T8313* rotary switch
MOVED	Position is changed	param 0<>	Bias value	For wheel of JogDial rotary switch -1.. 1; for wheel of frame-by-frame scrolling Shuttle -7..7 For keyboard *Panasonic WV-CU950* JogDial -1.. 1; Shuttle -6..6
		device <>	Type of used control mechanism *AXIS T8313*	0—wheel of rotary switch, 1—wheel of frame-by-frame scrolling

List of commands and parameters for the **TELEMETRY_EXT** object is presented in the following table:

Command—command description	Parameters	Description
"DRAW_FIGURE"—draw a figure on display of *BOSCH KBD-Digital or BOSCH KBD-Universal* telemetry panel	display<>	0x00—main display, 0x01—status display
	x1<>	Start coordinate on X axis (from 0 to 127 for main display, from 0 to 121 for status display)
	y1<>	Start coordinate on Y axis (from 0 to 239 for main display, from 0 to 31 for status display)
	x2<>	End coordinate on X axis (from 0 to 127 for main display, from 0 to 121 for status display)
	y2<>	End coordinate on Y axis (from 0 to 239 for main display, from 0 to 31 for status display)
	is_fill<>	0—do not fill the figure, 1—fill the figure
	is_set_pixels<>	0—remove figure from display, 1—draw figure
	figure<>	0—line, 1—rectangle
"PRINT_TEXT"—print text on display of *BOSCH KBD-Digital or BOSCH KBD-Universal* telemetry panel	display<>	0x00—main display, 0x01—status display
	x<>	Coordinate on X axis (from 0 to 127 for main display, from 0 to 121 for status display)
	y<>	Coordinate on Y axis (from 0 to 239 for main display, from 0 to 31 for status display)

	charset<>	Coding: 0—Latin 1—Cyrillic 2—Central European
	style<>	Style: 0—Normal 1—Semi-bold
	text<>	Test message
"PRINT_TEXT"—print text on display of *Panasonic WV-CU950* telemetry panel	y<>	0—display text on the first line 1—display text on the second line
	text<>	Displayed text of a line, maximum 20 characters
	flickering<>	Line consists of 6 characters determining parameters of text flashing: d1 d2 d3 d4 d5 d6 d1 determines period of flashing : 0—flashing disabled 1—period 0.25 sec, character is replaced by white space 2—period 0.5 sec, character is replaced by white space 3—period 0.75 sec, character is replaced by white space 4—period 1 sec, character is replaced by white space 5—period 0.25 sec, character is replaced by dark space 6—period 0.5 sec, character is replaced by dark space 7—period 0.75 sec, character is replaced by dark space 8—period 1 sec, character is replaced by dark space d2: 1—characters from 1 to 4 are flashing, 0—these character are not flashing d3: 1—characters from 5 to 8 are flashing, 0—these characters are not flashing d4: 1—characters from 9 to 12 are flashing, 0—thesecharacters are not flashing d5: 1—characters from 13 to 16 are flashing, 0—these characters are not flashing d6: 1—characters from 17 to 20 are flashing, 0—these characters are not flashing
"CLEAR_DISPLAY"—clear display of *BOSCH KBD-Digital or BOSCH KBD-Universal* telemetry panel. Reaction without parameters for the *Panasonic WV-CU950* telemetry panel	display<>	0x00—main display, 0x01—status display
"RELE_ON"—turn on the light on the *AXIS T8312* keyboard or *Panasonic WV-CU950* panel	rele<>	Code of key with the light, 12..16 for *AXIS T8312*. For Panasonic WV-CU950 see Installing and configuring security system components guide, section Features of Panasonic WV-CU950 control panel configuration and operation

"RELE_OFF"—turn off the light on the *AXIS T8312* keyboard or *Panasonic WV-CU950* panel	rele<>	Code of key with the light, 12..16
"RESET"—reset of *Panasonic WV-CU950* panel	type <>	0—instant reset 1—reset after 100 ms 2—reset after 200 ms 3—reset after 500 ms 4—reset after 1 s
"SET_ALARM"—set type of alarm signal of the *Panasonic WV-CU950* panel	audio_alarm<>	0—sound is off 1—simple single alarm signal 2—simple double alarm signal 3—simple triple alarm signal 4—single alarm signal lasting 0.1 sec 5—single alarm signal lasting 0.2 sec 6—single alarm signal lasting 0.3 sec 7—single alarm signal lasting 1 sec 8—simple single tone 9—simple double tone A—simple triple tone B—single signal lasting 0.1 sec C—single signal lasting 0.2 sec D—single signal lasting 0.3 sec E—single signal lasting 1 sec F—alarm signal

JOYSTICK Control device

The **JOYSTICK** object corresponds to the **Control device** object.

The **JOYSTICK** object sends the events presented in the table. The procedure is started when the corresponding event occurs.

Events	Parameter	Parameter description
"KEY_PRESSED"—The key is pressed	button<>	Key code
	cam<>	Camera ID
	param0<>	Name of the Remote Client to which the device is connected

TIME_ZONE Time zone

The **TIME_ZONE** object corresponds to the **Time zone** system object.

The **TIME_ZONE** object sends events presented in the table. Procedure is started when the corresponding event occurs.

Event	Description
ACTIVATE	Start
DEACTIVATE	End

List of commands and parameters for the **TIME_ZONE** object is presented in the following table:

Command—command description	Parameters	Description
"SETUP"—set parameters for time zone	name<>	Object name
	flags<>	Flags

Properties of the **TIME_ZONE** object are shown in the table.

Properties of the TIME_ZONE object	Description of properties
ID<>	Object ID
PARENT_ID<>	Parent object ID

The **TIME_ZONE** object can be in the following states:

State of the TIME_ZONE object	Description
"ACTIVATE"	Active
"INACTIVE"	Inactive

ARCH Backup archive

The **ARCH** object corresponds to the **Backup archive** system object.

The **ARCH** object sends events presented in the table. Procedure is started when the corresponding event occurs.

Events	Description	Comment
ACTIVE	Backup archive is active	Event is generated when a list of cameras, the video from which is archived, corresponds to the list of Backup archive configuration
INACTIVE	Backup archive is inactive	Event is generated when archiving via Backup archive is not performed.
ACTIVE_ PART	Partial operation of Back up archive	Event is generated when archiving is enabled not for all cameras specified in the list of Backup archive.

FAILOVER Failover service

The **FAILOVER** object corresponds to the **Failover service** object.

Events from the **FAILOVER** object are given in the table. Procedure is started when the corresponding event occurs.

Event	Description
START	Objects are moved to a backup Server
STOP	Objects are returned to the main Server

The list of commands and parameters for the **FAILOVER** object is presented in the table below:

Command—command description	Comment
FORCED_START—forced transfer of the main Server configuration to the backup Server	The reverse configuration transfer is performed using the FORCED_STOP command or upon restarting/reconnecting the main Server
FORCED_STOP—forced transfer of the backup Server configuration to the main Server	

OPERATORPROTOCOL Operator protocol

The **OPERATORPROTOCOL** object corresponds to the **Operator protocol** system object.

Find events from the **OPERATORPROTOCOL** object in the table. Procedures are started when the corresponding event occurs.

Event	Event description	Event parameters
ACTIVATE_LEFT	Operator left-clicked the event cell in the **Operator protocol**	
ACTIVATE_RIGHT	Operator right-clicked the event cell in the **Operator protocol**	
POSTPONE_PRESSED	Operator clicked the **Delay** button	
CREATE_REPORT	Operator clicked the **Create** button on the **Create report** tab	The user_id<> parameter contains the user ID. The initial_date<> and final_date<> parameters specify the initial and final dates selected in the interface
RESPONSE_ALARM	Operator clicked the **Alarm situation** button	objtype<>—Object type objid<>—Object ID action<>—Name of the event in the database alarm_time<>—Time when the alarm occurred
RESPONSE_SUSPECT	Operator clicked the **Suspicious situation** button	
RESPONSE_FALSE	Operator clicked the **False alarm** button	
ACTIVATE_EVENT	Focusing on the event: click the event in the interface or jump to the required event using the keyboard	

The list of commands and parameters for the **OPERATORPROTOCOL** object is given in the table.

Command—command description	**Parameters**	**Parameter description**
DEL_ALARM—delete an alarm	objtype<>	Object type (for example, CAM, GRELE, and so on)
	objid<>	Object ID
	options<>	Possible values: • first—delete first alarm • last—delete last alarm • all or blank—delete all alarms
HIDE_BUTTON – hide the buttons used to assign status to an event	button<>	Names of the buttons separated by comma: • alarm—Alarm situation • suspicious—Suspicious situation • false—False alarm Example of setting a parameter: `button<alarm,suspicious,false>`
	hide<>	1—hide the buttons listed in the button parameter 0—show the buttons listed in the button parameter
	objtype<>	Object type
	objaction<>	Event type

	objid<>	Object ID

EVENT_VIEWER Event Viewer

The **EVENT_VIEWER** object corresponds to the **Event Viewer** system object.

The **EVENT_VIEWER** object sends events given in the table. Procedure is started when the corresponding event occurs.

Description of events of the **EVENT_VIEWER** object.

Events	Description
SHOW_ON_MAP	The operator selected the "Display on the map" command
SHOW_VIDEO	The operator selected the "Display video" command
SHOW_REPORT	The operator selected the "Show report" command
CREATE_REPORT	Generated if the GenerateEventInsteadOfReport registry key is set to 1 and the operator selected the "Show report" command. The report does not open. See also Registry keys reference guide

List of commands and parameters for the **Event Viewer** object is presented in the following table:

Command—command description	Parameters	Description of parameters
UPDATE_VIEW—set general background and/or text color in the Event Viewer interface window	bk_color <>	General background color in x16 format
	defclr<>	General text color in x16 format
"SET_FILTERS"—activate the Event Viewer filter	filter0<>, filter1<>, filter2<>, etc.	The name of the filter created in the Event Viewer. If you need to specify several filters, then the parameter numeration (0, 1, 2, etc.) must go in order starting with filter0<>. Note that the numbers should not be skipped. See Examples of scripts with Event Viewer

GATE Videogate

The **GATE** object corresponds to the **Videogate** system object.

The **GATE** object sends events presented in the table. Procedure is started when the corresponding event occurs.

Events	Description	Comment
GATE_LOW_FPS	Input speed on the gate is reduced	
ACTIVE	Gate is active	Event is generated when the list of working cameras corresponds to the list of the Videogate configuration
INACTIVE	Gate is inactive	Event is generated when there are no requests for video streams through the Videogate
ACTIVE_PART	Partial operation of gate	Event is generated when the number of working cameras is less than in the Videogate list

The list of commands and parameters for the **GATE** object is presented in the table.

Command—command description	Parameters	Parameter description	Features
START_VIDEO—enable camera video stream and start writing to the archive	cam<>	Identifier of the camera by which it is necessary to start or stop recording	The commands work even if the Monitor doesn't display the selected camera. The commands work if constant recording and active camera recording is enabled in the Videogate—see Configuring the recording to the Videogate archive
STOP_VIDEO—stop camera video stream and stop writing to the archive			

CAM_VMDA_DETECTOR VMDA detection

The **CAM_VMDA_DETECTOR** object corresponds to the **VMDA detection** system object.

The **CAM_VMDA_DETECTOR** object sends events presented in the table. Procedure is started when the corresponding event occurs.

Event	Description	Parameter	Parameter description
ALARM	Alarm	native_type<>	To enable this parameter, set the following registry keys: VMDA.determineNoise, VMDA.determineGivenTaken, VMDA.determineHumanCar (see Registry keys reference guide). Available values: -1—unknown object type (initial state) 0—other 1—human or group of people (depending on the native_value<> parameter: if 1, human; if >1, group of people) 2—car 3—noise 4—object carried into the area 5—object carried out of the area
		native_value<>	To enable this parameter, set the following registry keys: VMDA.determineNoise, VMDA.determineGivenTaken, VMDA.determineHumanCar (see Registry keys reference guide). People counter for the "human" object type. Allows determining the quantity of people in the group. For other object types it is equal to -1
		param0<>	String value containing the event parameters from the VMDA detection
ALARM_END	End of alarm		
ARMED	VMDA detection is armed		
DISARMED	VMDA detection is disarmed		

List of commands and parameters for the **CAM_VMDA_DETECTOR** object is presented in the following table:

Command—command description	Parameters	Description
ARM—arm detection	-	-
DISARM—disarm detection	-	-

Note

If cameras are connected using ONVIF Server, the events from the **VMDA detection** and other smart detection tools will be transferred as events from embedded detection tools—see CAM_IP_DETECTOR Embedded detection.

TITLEVIEWER Captions search

The **TITLEVIEWER** object corresponds to the **Captions search** system object.

The events given in the table come from the **TITLEVIEWER** object. Procedures are started when the corresponding event occurs.

Event	Event description	Parameters	Parameters description	Comment
GO_VIDEO	Video request	<cam>	The ID of the camera where the captions were found	The event is generated when left double-clicking the search result line
		<date>	Date	
		<time>	Time	

PERSON User

The **PERSON** object corresponds to the **User** system object.

The events presented in the table come from the **PERSON** object. Procedures are started when the corresponding event occurs.

Event	Description
REGISTERED	User logs in to the system
UNREGISTERED	User logs out of the system

CAM_FACECAPTURE Face Detection

The **CAM_FACECAPTURE** object corresponds to the **Face Detection** system object.

The **CAM_FACECAPTURE** object sends the events presented in the table below. The procedure is started when the corresponding event occurs.

Events	Events description
FACE_DETECTED	Face is captured
FACE_LEAVE	Face is lost

The list of parameters for the **CAM_FACECAPTURE** object is presented in the table:

Parameters	Parameters description
owner	The name of the server where the face was captured/lost
fraction	The millisecond when the face was captured/lost
module	The module where the face is captured
date	The date when the face was captured/lost
guid_pk	The event ID (generated randomly for each event)
core_global	Distribute to all cores (notify all)
guid	The captured/lost face ID (generated randomly for each event)
time	The time when the face was captured/lost
param0	Same as the **guid** parameter. Used to display the information in the "Add. info" column of the Event Viewer

IPSTORAGE Edge storage

The **IPSTORAGE** object corresponds to the **Edge storage** system object.

The list of parameters and commands for the **IPSTORAGE** object is presented in the table:

Command—description	**Parameter**	**Parameter description**	**Comment**
IMPORT—import of the missing part of the archive for the specified period	cam<>	Camera ID	The command is used if automatic import for the Edge storage was not successful. *Note. If import is performed at the time of sending the reaction, the command will not be executed. To abort the current import task, first send the UPDATE_TIME command*
	datetime_from<>	Date and time to start import from in the following format: <DD-MM-YY HH:MM:SS>	
	datetime_to<>	Date and time to end import on in the following format: <DD-MM-YY HH:MM:SS>	
UPDATE_TIME—stop synchronization and set the time of the last import from the edge storage in the Settings.xml file	cam<>	Camera ID	The command is used to stop the current import task and execute the IMPORT command to synchronize the specified archive period
	datetime<>	Date and time of the last synchronization to be set in the Settings.xml file	

CAM_TITLE Captioner

The **CAM_TITLE** object corresponds to the **Captioner** system object.

The list of commands and parameters for the **CAM_TITLE** object is given in the table:

Command—command description	Comment
"REINDEX"—run the captions database update	Captions database re-indexation is run at any value of the _id_ parameter in the DoReact( ) command

TELEGRAM Telegram bot

The **TELEGRAM** object corresponds to the **Telegram bot** system object.

The **TELEGRAM** object sends events presented in the table. The procedure is started when the corresponding event occurs.

Event	Description	Comment
ERROR	Message sending error	The error<> parameter contains a test description of the error

The list of commands and parameters for the **TELEGRAM** object is given in the table:

Command—command description	Parameters	Description
SEND—send	text<>	Message text
	chat_id<>	Chat identifier
	bot_id<>	Bot identifier
	longtitude<>	Geolocation longtitude
	latitude<>	Geolocation latitude
	address<>	Geolocation text address
SENDPHOTO—send photo	photo<>	Full path to the image file
	caption<>	File caption
	chat_id<>	Chat identifier
	bot_id<>	Bot identifier
	longtitude<>	Geolocation longtitude
	latitude<>	Geolocation latitude
	address<>	Geolocation text address

CAM_IP_DETECTOR Embedded detection

The **CAM_IP_DETECTOR** object corresponds to the **Embedded detection** system object.

The **CAM_IP_DETECTOR** object sends events given in the table. Procedure is started when the corresponding event occurs.

Events	Events description	Comment
DETECTED	Event	The event occurs when metadata is received from embedded detection tools. For example, this event occurs when receiving the data on body temperature from a thermal camera, and so on. The event also occurs if the data from detection tools is transferred using Onvif. The param0<> parameter has a string value containing the event parameters. The transferred parameters depend on the embedded detection tool. If you use Onvif, you can configure the contents of the parameter on the **Event settings** tab of the **ONVIF-Server** object (see Filtering the ONVIF-Server events)

Examples of events from embedded detection tools:

Example 1

```
// Event from a thermal camera
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
}
;>
```

Example 2

```
// Event from VMDA detection
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<12:22:30>,_timestamp<7520749>,
fraction<465>,date<30-03-21>,owner<DESKTOP-339SH3U>,module<video.run>
```

SIP_TERMINAL SIP-terminal

The **SIP_TERMINAL** object corresponds to the **SIP-terminal** system object.

The **SIP_TERMINAL** object sends events given in the table. Procedure is started when the corresponding event occurs.

Event	Description	Contents of the param0<> parameter displayed in the Add. info field in the Event Viewer
CALL_END	Call end	Number of the subscriber who called
CALL_END_OPERATOR	Operator call end	Numbers of the subscribers and call duration. For example, if the parameter takes the "903 to 906 (01:04)" value, it means that the subscriber 903 called the subscriber 906, and the call lasted 1 minute and 4 seconds
CALL_END_DEVICE	Device call end	
CALL_END_VIRTUAL	Special number call end	
CALL_BEGIN	Call start	Numbers of the subscribers: the subscriber who is calling and the subscriber who is being called
CALL_TRYING	Call attempt	
CALL_BEGIN_VIRTUAL	Start of special number call	
CALL_TRYING_VIRTUAL	Special number call attempt	

List of commands and parameters for the SIP_TERMINAL object:

Command—command description	Parameters	Parameters description
END_ALL_CALLS—end all calls on the specified terminal (regardless of whether the connection is established)	-	-

INC_MANAGER Incident manager

The INC_MANAGER object corresponds to the **Incident manager** system object.

The INC_MANAGER object sends events given in the table. Procedures are started when the corresponding event occurs.

Events	Events description	Paremeters	Parameters description	Comment
CLOSE_CLICK	Click the **Close** button in the interface			The event is generated when an incident is closed without being processed by the operator in the **Incident manager** interface
CLOSE_ALL_CLICK	Click the **Close all** button in the interface			The event is generated when all incidents are closed without being processed by the operator in the **Incident manager** interface
SELECT	Click an incident in the interface			The event is generated if the operator left-clicks or right-clicks the incident in the **Incident manager** interface
ACTIVATE_EVENT	The operator selected the event (click the event with the mouse)	alarm_time<>	Time when the event occurred	
		event_guid<>	Event ID (generated randomly for each event)	
		objtype<>	Object type (for example, CAM, GRELE, and so on)	
		action<>	Action type (for example, MD_START, DISARM, and so on)	

INC_SERVER Incident server

The INC_SERVER object corresponds to the **Incident server** system object.

The INC_SERVER object sends events presented in the table. The procedure is started when the corresponding event occurs.

Events	**Events description**	**Comment**
EVENT	The event (incident) is taken into processing in the **Incident manager** or is being processed by the operator	The event is generated: 1. when the operator takes the event into processing; 2. at each step of the event processing. The **serializeBase64** parameter of the event contains JSON with the detailed information about the processed event, including the steps performed by the operator. You can enable full logging of all event parameters using the registry key **inc_server_send_full_event** (see Axxon PSIM base version). When inc_server_send_full_event=1: • the **serializeBase64** parameter will contain an encoded string with the **SourceMsgBase64** field, which contains the encoded event string. This event contains the **full_event_base64** parameter with the full parameters of the source event; • if the event is escalated, an additional parameter **inc_server_action<escalated>** will also be added. Please note that if you enable full parameter logging, it will increase the size of the event database (by default, **Incident server** events are stored in the *Axxon PSIM* database in the PROTOCOL_INC_SERVER table)

The list of commands and parameters for the INC_SERVER object is presented in the table.

Command—description of the command	**Parameters**	**Parameters description**	**Comment**
UPDATE_STATUS—change the status of the event (incident) in the **Incident manager**	pks<>	Array of event identifiers	Parameters are filters and the presence of at least one parameter is mandatory. Parameter values can be specified with a delimiter. This means that one OR the other value will be selected. Example: objids<1|2>
	objtypes<>	Objects types	
	objids<>	Objects identifiers	
	actions<>	Actions	
	status<>	Event status: 0—Waiting to be processed 1—Processing 2—Suspended 3—Completed	
UPDATE_ESCALATE_STATUS—change the status of the event (incident) escalation in the **Incident manager**	escalated<>	Event escalation status: 0—Waiting to be processed (not escalated) 1—Escalated	
	pks<>, objtypes<>, objids<>, actions<> are the same as for UPDATE_STATUS		

DIALOG Operator query panel

The **DIALOG** object corresponds to the **Operator query panel** system object:

List of commands and parameters for the **DIALOG** object is presented in the table:

Command—command description	**Parameters**	**Description**
"SETUP"—set up the operator query panel	x<>	Coordinate of left top corner (0–100)
	y<>	Coordinate of left top corner (0–100)
	allow_move<>	0—forbid moving, 1—allow moving
"RUN"—show the operator query panel	-	-
"RUN_MODAL"—run the operator query panel in modal mode	-	-
"CLOSE"—close last opened operator query panel	-	-
"CLOSE_ALL"—close all opened operator query panels	-	-

MMS Mail Message Service

The **MMS** object corresponds to the **Mail Message Service** system object.

The **MMS** object sends events presented in the table. Procedure is started when the corresponding event occurs.

Event	Description
"SET_CONNECTIONS"	List of available connections

List of commands and parameters for the MMS object is given in the table:

Command—command description	Parameters	Description
"SETUP"—settings for the mail message service	smtp<>	Address of the SMTP server
	connection<>	Type of connection
	smtp_username<>	Username
	smtp_password<>	Password
	port<>	Port number
	flags<>	Flags
	name <>	Object name
"GET_CONNECTIONS"—get the list of available connections	-	-

Properties of the **MMS** object are shown in the table:

Properties of the MMS object	Description
ID<>	Object ID
PARENT_ID<>	Parent object ID

MAIL_MESSAGE Mail message

The **MAIL_MESSAGE** object corresponds to the **Mail message** system object.

The **MAIL_MESSAGE** object sends events presented in the table. Procedure is started when the corresponding event occurs.

Event	Description
SEND_ERROR	Message sending error
SENT	Message is sent

List of commands and parameters for the **MAIL_MESSAGE** object is given in the table:

Command—command description	Parameters	Description
"SETUP"—settings for mail message	from<>	Sender's address
	to<>	Recipient's address
	cc<>	Copies
	subject<>	Message subject
	body<>	Message body
	attachments<>	Attachments. If several files are attached, their addresses are separated with a semicolon
	flags<>	Flags
	name<>	Object name
	pack<>	Method of attachments packing
	is_body_html<>	Specifies if HTML markup must be applied when sending. Possible values: 1 or 0
	inline<>	Specifies if attachments are only shown in the message text (value of 1) or both in the text and in the "Attachments" section (value of 0)
"SEND"—send mail message	-	-
"SEND_RAW"—send email with parameters	same as for the SETUP command	see Examples of scripts in the JScript language

Properties of the **MAIL_MESSAGE** object are shown in the table.

Properties of the MAIL_MESSAGE object	Description
ID<>	Object ID
PARENT_ID<>	Parent object ID

VMS Voice Message Service

The **VMS** object corresponds to the **Voice Message Service** system object.

List of commands and parameters for the **VMS** object is presented in the table:

Command—command description	Parameters	Description of parameters
"SEND"—send message.	modem<>	Name of device
	pulse<>	Type of dialing (0—tonal, 1—pulse)
	name<>	Object name
	redial_attempts<>	Number of call attempts
	redial_delay<>	Pause between call attempts
	waitfordialtone<>	Waiting for line signal (0—no, 1—yes)
	flags<>	Flags

Properties of the **VMS** object are shown in the table:

Properties of the VMS object	Description of properties
ID<>	Object ID
PARENT_ID<>	Parent object ID

GRELE Relay

The **GRELE** object corresponds to the **Relay** system object.

The **GRELE** object sends events presented in the table. Procedure is started when the corresponding event occurs.

Event	Description
OFF	Relay is off
ON	Relay is on
SIGNAL_LOST	Connection lost

List of commands and parameters for the **GRELE** object is presented in the table:

Command—command description	Parameters	Description
"ON"—enable relay	-	-
"OFF"—disable relay	-	-
"SETUP"—settings for relay	chan <>	Output number (0–15)
	flags<>	Flags
	name<>	Object name

Properties of the **GRELE** object are shown in the table.

Properties of the GRELE object	Description of properties
ID<>	Object ID
PARENT_ID<>	Parent object ID
REGION_ID<>	Region ID

The **GRELE** object can be in the following states:

State of the GRELE object	State description
"ON"	Relay is on
"OFF"	Relay is off
"DETACHED_ON"	Connection lost, relay was on
"DETACHED_OFF"	Connection lost, relay was off

GRAY Sensor

The **GRAY** object corresponds to the **Sensor** system object.

The **GRAY** object sends events presented in the table. Procedure is started when the corresponding event occurs.

Event	Event description
ALARM	Alarm. This event is received when the sensor is opened or closed (it depends on object settings) if sensor is armed. If sensor is disarmed, then Sensor opened and Sensor closed events are received respectively
ARM	Sensor is armed
CONFIRM	Alarm received
DISARM	Sensor is disarmed
NOT_VALID_STATE	Zone is not ready
OFF	Sensor is opened. This event is received when the sensor is opened if the sensor is disarmed
ON	Sensor is closed. This event is received when the sensor is closed if the sensor is disarmed
SIGNAL_LOST	Connection with the sensor is lost

List of commands and parameters for the **GRAY** object is presented in the table:

Command – command description	Parameters	Description
"ARM"—arm the sensor	-	-
"DISARM"—disarm the sensor	-	-
"CONFIRM"—confirm an alarm	-	-
"SETUP"—settings for the sensor	chan<>	Output number (0–15)
	flags<>	Flags
	name<>	Object name
	type<>	Type of sensor object (0—on closing, 1—on opening)

Properties of the **GRAY** object are shown in the table.

Properties of the GRAY object	Description of properties
ID<>	Object ID
PARENT_ID<>	Parent object ID
REGION_ID<>	Region ID

The **GRAY** object can be in the following states:

State of the GRAY object	State description
"ARMED"	Sensor is armed
"DISARME""	Sensor is disarmed
"ALARMED"	Alarm
"CONFIRMED"	Alarm confirmed
"DISARMED_ALARM"	Not ready
"DETACHED_ARMED"	Connection lost when the sensor was armed
"DETACHED_DISARM"	Connection lost when the sensor was disarmed

“OFF”	Normal

VNS Voice notification service

The **VNS** object corresponds to the **Voice notification service** system object.

List of commands and parameters for the **VNS** object is presented in the table:

Command—command description	**Parameters**	**Description**
"SETUP"—settings of the voice notification service	card <>	Name of sound device. **Note**. Card name must correspond to the name specified in the settings of the sound card of the **Voice notification service**
	level <>	Level of signal. The value of the parameter varies from 0 to 15. By default, it is 8
	channel <>	Set of sound channels. Possible values of the parameter: 0—no sound channel; 1—left playback channel; 2—right playback channel; 3—left and right playback channels (both channels)
	flags <>	Flags
	ip <>	IP-address of network device
	name <>	Object name
	pass <>	Password
	user <>	Username
"PLAY"—play audio file	file <>	Full path to the audio file in .wav format (indicating the name of the file being played. For example: C:\Program Files (x86)\Axxon PSIM\Wav\cam_alarm_1.wav). **Note**. If only file name is specified, the default path to the file will be taken from the InstallPath registry key in the HKEY_LOCAL_MACHINE\SOFTWARE\AxxonSoft\PSIM section (HKEY_LOCAL_MACHINE \Software\Wow6432Node\AxxonSoft\PSIM for 64-bits system), in value of the InstallPath parameter. In this parameter, it is possible to play several audio files using the "+" operation
"STOP"—stop playing audio file	-	-

Properties of the **VNS** object are shown in the table.

Properties of the VNS object	**Description of properties**
ID<>	Object ID

PARENT_ID<>	Parent object ID

SMS Short Message Service

The **SMS** object corresponds to the **Short Message Service** system object.

The **SMS** object sends events presented in the table. Procedure is started when the corresponding event occurs.

Event	Event description	Comment
RECEIVE	Message is received	If the event doesn't occur when a message is received on the modem, you must use the ProcessFromSim registry key (see Registry keys reference guide). The message<> message parameter contains the text of the sent message. The parameter phone<> contains the phone number from which the message was sent, in the +7XXXXXXXXXX format

List of commands and parameters for the **SMS** object is presented in the table:

Command—command description	Parameters	Description of parameters
"SETUP"—settings of short message service	device<>	SMS device
	flags<>	Flags
	message<>	Message text
	name<>	Object name
	phone<>	Phone number

Properties of the **SMS** object are shown in the table.

Properties of the SMS object	Description of properties
ID<>	Object ID
PARENT_ID<>	Parent object ID

SSS_WATCHDOG System restart service

The **SSS_WATCHDOG** object corresponds to the **System restart service** system object.

The **SSS_WATCHDOG** object sends events presented in the table. Procedure is started when the corresponding event occurs.

Event	Description
RESTART_EXCEEDED	Number of module restarts is exceeded
RESTART_PROCESS	Module restart

List of commands and parameters for the **SSS_WATCHDOG** object is presented in the table:

Command—command description	Parameters	Description
"SETUP"—set parameters for the system restart service	name<>	Object name
	flags<>	Flags
	restart_period<>	Restart period
	restart_times<>	Maximal number of restarts for the specified period
	timeout<>	Response time
	usb_wd_control<>	Connecting AxxonSoft USB Watchdog

Properties of the **SSS_WATCHDOG** object are shown in the table:

Properties of the SSS_WATCHDOG object	Description of properties
ID<>	Object ID
PARENT_ID<>	Parent object ID

BACNET BacNet

The **BACNET** object corresponds to the **BacNet** system object.

The **BACNET** object sends events presented in the table. Procedures are started when the corresponding event occurs.

Event	Event description
ERROR	Error message received
EVENT_OCCURES	Message confirmation
WRITE_OCCURES	Recording confirmation
WRITE_RESULT	Recording result

The list of commands and parameters for the **BACNET** object is presented in the table:

Command—description	Parameters	Parameters description

WRITE—send a value to the BACnet device	bacnet_application_tag<>	Data type. Possible values:
		`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<>	Parameter value

	bacnet_objtype<>	Object type:
		`ANALOG INPUT = 0`
		`ANALOG OUTPUT = 1`
		`ANALOG VALUE = 2`
		`BINARY INPUT = 3`
		`BINARY OUTPUT = 4`
		`BINARY VALUE = 5`

	bacnet_instance<>	BACnet unique global device identifier
	bacnet_property_id<>	Property ID
	bacnet_device_id<>	BACnet device ID in the system
EVENT—send a message to the BACnet device	event_type<>	Event type
	from_state<>	Change state from
	to_state<>	Change state to
	message_text<>	Event text

Description of the object model in Axxon PSIM

The Core object and its built-in methods

All methods of the Core object can be used in **Scripts**. The following methods can also be used in **Programs**:

- Sleep
- DoReact
- DoReactGlobal
- NotifyEvent
- NotifyEventGlobal

The rest of the Core object methods are not used in **Programs**.

The MsgObject and Event objects and their built-in methods and properties

All methods of the MsgObject and Event objects are used only in **Scripts**.

The Core object and its built-in methods

The Core object

The **Core** object is a core of the system, a global static object that implements the methods used to control the state and monitor the system objects of *Axxon PSIM*. The methods of the **Core** object allow you to receive information about the registered system objects, generate reactions for them and change their states. The **Core** object implements additional methods used to pause script execution, debug scripts, create and call global variables.

The **Core** object is not a prototype, thus you cannot create other objects on its basis (it cannot be used as a template). All methods of the **Core** object are static. Thus, you can call the methods of the **Core** object directly from the script without referring to the **Core** obje ct itself.

The GetObject methods

The GetObjectName method

The GetObjectName method returns the name of the object that it was given during registration in the program.

Syntax for method invocation:

```
function GetObjectName(objtype : String, id : String) : String
```

Method arguments:

1. **objtype** is a required argument. It specifies the system type of the object which name you want to get. Possible values: String type, range is limited by object types registered in the system.
2. **id** is a required argument. It corresponds to the identification (registration) number of the object specified by the **objtype** argument. Possible values: String type, the range is limited by the identification numbers of objects of the specified type registered in the system.

Example. On alarm in any sensor, open the information window with the following text: "Alarm in the <alarmed sensor name>. Sensor connected to the Server <name of the section to which the sensor is connected>".

> **Note**
>
> You must create the information dialog window beforehand using the *Arpedit.exe* utility and save it in the test.dlg file in the <*Axxon PSIM* installation directory>\Program folder.

```
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","Alarm in the '" + grayname + "' sensor connected to the '" + compname + "'
server.");
}
```

The GetObjectParam method

The GetObjectParam method returns the value of the specified parameter of the system object at the moment of method invocation.

Syntax for method invocation:

```
function GetObjectParam(objtype : String, id : String, param : String) : String
```

Method arguments:

1. **objtype** is a required argument. It specifies the type of system object for which you want to return the value of the specified parameter. Possible values: String type, range is limited by object types registered in the system.
2. **id** is a required argument. It corresponds to the identification (registration) number of the object specified by the **objtype** argument. Possible values: String type, the range is limited by the identification numbers of objects of the specified type registered in the system.
3. **param** is a required argument. It corresponds to the name of the parameter the value of which you want to return. Possible values: String type, range is limited by the parameters available for the given object.

Example. See the example in The SetObjectParam method.

The GetObjectParentId method

The GetObjectParentId method returns the identification (registration) number of the parent object of the specified object.

Syntax for method invocation:

```
function GetObjectParentId(objtype : String, id : String, parent : String) : String
```

Method arguments:

1. **objtype** is a required argument. It specifies the type of the system object, for which you want to return the type of parent object. Possible values: String type, range is limited by object types registered in the system.
2. **id** is a required argument. It corresponds to the identification (registration) number of the object specified by the **objtype** argument. Possible values: String type, range is limited by the identification numbers of objects of the specified type registered in the system.
3. **parent** is a required argument. It specifies the type of the system object that is the parent object (main object in the system hierarchy of objects) of the object specified by the **objtype** argument. Possible values: String type, range is limited by object types registered in the system.

Example. If a camera turns off or stops transmitting a video signal, send an email message registered in *Axxon PSIM* under number 1. The message must include the subject: "Warning! Camera turned off" and, in the message body, the number of the camera and of the server it is connected to.

> **Note**
>
> The *Short Messages Service* must be installed and work properly.

```
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.com>,to<***@mail.com>,body<Camera disabling "+cam_id+" on
the Server"+parent_comp_id+">,parent_id<1>,subject<Attention! Camera disabling>,name<Message 1>,objname<Message
1>");
 DoReactStr("MAIL_MESSAGE", "1", "SEND", "");
}
```

The GetObjectState method

The GetObjectState method returns the state of the object at the moment of method invocation.

Syntax for method invocation:

```
function GetObjectState(objtype : String, id : String) : String
```

Method arguments:

1. **objtype** is a required argument. It specifies the type of the system object which state you want to get. Possible values: String type, range is limited by object types registered in the system.
2. **id** is a required argument. It corresponds to the identification (registration) number of the object specified by the **objtype** argument. Possible values: String type, the range is limited by the identification numbers of objects of the specified type registered in the system.

Example. When relay 1 is enabled (for example, on pressing the button connected to the relay 1), arm sensor 1. The next time the relay 1 is enabled, disarm sensor 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

Some object types can have several states at the same time. For example: ATTACHED|DISARMED or ATTACHED|DISARMED|RECORDER_ON|RECORDING.

The GetObjectParentType method

The GetObjectParentType method returns the type of the parent object for the specified object according to the system object hierarchy.

Syntax for method invocation:

```
function GetObjectParentType (objtype : String) : String
```

Method arguments:

1. **objtype** is a required argument. It corresponds to the type of system object for which you want to return the type of the parent object. Possible values: String type, range is limited by object types allowed in the system.

> **Note**
>
> In the hierarchy of system objects, the Main object is the highest level object. This object is the parent object for all objects of the Computer type, Screen type and others.

Example. On macro 1, display in the debug window the names of four objects, starting from the detection zone, in the order of the hierarchy provided by *Axxon PSIM*.

```
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);
 }
}
```

The GetObjectIdByParam method

The GetObjectIdByParam method allows you to get the ID of an object, some parameter of which is equal to the specified value. If there are several such objects, the ID of the first found object is returned. If no such objects are found, 0 is returned.

Syntax for method invocation:

```
function GetObjectIdByParam (obj_type : String, obj_param : String, param_value : String)
```

Method arguments:

1. **obj_type** is a required argument. It specifies the type of system object, the ID of which you want to get.
2. **obj_param** is a required argument. It specifies the name of the parameter in the database, by the value of which you want to search for the object.
3. **param_value** is a required argument. It specifies the required value of the object parameter.

Example. Find cameras with a black-and-white video image and set the **Color** parameter to 1 for them.

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

{
 var id = GetObjectIdByParam("CAM","color","0"); //get the first object ID
 while (id){ //while exist the Camera objects from which black-and-white image is received
  SetObjectParam ("CAM", id, "color", "1"); //change the Color parameter for found object
  id = GetObjectIdByParam("CAM","color","0"); //get the next object ID
 }
}
```

The GetObjectChildIds method

The GetObjectChildIds method returns the identification (registration) numbers of the objects of the specified type that are below the specified object in the object hierarchy.

Syntax for method invocation:

```
function GetObjectParentId(parent : String, id : String, objtype : String) : String[]
```

Method arguments:

1. **parent** is a required argument. It specifies the type of the system object which child objects you want to find out. Possible values: String type, range is limited by object types registered in the system.
2. **id** is a required argument. It corresponds to the identification (registration) number of the object specified by the **parent** argument. Possible values: String type, range is limited by the identification numbers of objects of the specified type registered in the system.
3. **objtype** is a required argument. It specifies the type of the system object which is a child of the type specified by the **parent** argument. Possible values: String type, range is limited by object types registered in the system.

Example. On macro 1, arm all cameras on the WS2 computer.

```
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","");
 }
}
```

The DoReact methods

The DoReact method

The DoReact method generates the reactions of the system objects. The method sends the reaction to the specified object. The reaction is transferred directly to the kernel on which the object is registered, and not to the entire system. In the DoReact method, the reaction is specified by the **MsgObject** object.

Syntax for method invocation:

```
function DoReact(msgevent : MsgObject)
```

Method arguments:

1. **msgevent** is a required argument. It specifies the reaction sent to the specified object. Possible values: the **MsgObject** objects created earlier in the script.

> **Note**
>
> Two types of system messages are available *Axxon PSIM:* events and reactions. The events usually contain some information and are used as notifications sent to all *Axxon PSIM* kernels connected to each other during the architecture configuration. The reactions are commands sent to specific objects. A reaction is transmitted only to the kernel on which the required object is registered, and not to the entire system. The DoReactStr and DoReact methods are used to generate reactions. The NotifyEventStr and NotifyEvent methods are used to generate events.

Example. When relay 1 closes, close relays 2 and 3. When relay 1 opens, open relay 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);
 }
}
```

The DoReactStr method

The DoReactStr method generates the reactions of the system objects. The method sends the reaction to the specified object. The reaction is transferred directly to the kernel on which the object is registered, and not to the entire system. In the DoReactStr method, the reaction is specified by a group of String arguments.

Syntax for method invocation:

```
function DoReactStr(objtype : String, id : String, action : String, param<value> [, param<value>] : String)
```

Method arguments:

1. **objtype** is a required argument. It corresponds to the type of system object for which you want to generate a reaction. Possible values: String type, range is limited by object types registered in the system.
2. **id** is a required argument. It corresponds to the identification (registration) number of the object specified by the **objtype** argument. Possible values: String type, range is limited by the identification numbers of objects of the specified type registered in the system.
3. **action** is a required argument. It specifies the reaction that you want to generate. Possible values: String type, range is limited by the reactions available for the object of the specified type.
4. **param<value>** is a required argument. You can specify several arguments of this type. It corresponds to the parameter(s) of the system object reaction.

Syntax for setting a value to a parameter corresponds to a string:

"param<value>", where

param is a name of the parameter;

value is a value of the parameter.

Syntax for setting a value to several parameters corresponds to a string:

"param1<value1>,param2<value2>…"

Elements of the list are separated by commas without spaces. If no parameter needs to be specified, an empty string is used:

```
DoReactStr("CAM","1","REC","");
```

Possible values of the **param** argument: String type, range is limited by the available parameters of the specified reaction. Possible values of the **value** argument: String type, range depends on the parameter that you want to set.

For all reactions you can specify delay of reaction execution using the delay<> parameter. Delay is specified in seconds.

> **Note**
>
> Two types of system messages are available in *Axxon PSIM*: events and reactions. The events usually contain some information and are used as notifications sent to all *Axxon PSIM* kernels connected to each other during the architecture configuration. The reactions are commands sent to specific objects. A reaction is transmitted only to the kernel on which the required object is registered, and not to the entire system. The DoReactStr and DoReact methods are used to generate reactions. The NotifyEventStr and NotifyEvent methods are used to generate events.

Example 1. When an alarm is received from a camera, switch Monitor 1 to single Surveillance window mode (one-fold) and show the video from the alarmed camera in this window.

```
 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>");
}
```

Example 2. When alarm from a camera is completed, the recording must continue for five second and after this time the recording must stop (analogue of Post-record mode).

```
if (Event.SourceType == "CAM" && Event.Action == "MD_STOP")
{
 var camid = Event.SourceId;
 DoReactStr("CAM",camid,"REC_STOP","delay<5>");
}
```

Example 3. On macros 1, enable telemetry control using mouse on the camera 4 displayed in the monitor 10. On macros 2, disable it.

```
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>");
}
```

The DoReactSetup method

The DoReactSetup method is used to temporary change the parameters of the system object. The method changes only the specified parameters of the object, leaving other parameters unchanged.

Syntax for method invocation:

```
function DoReactSetup (objtype : String, id : String, param<value> [, param<value>] : String )
```

Method arguments:

1. **objtype** is a required argument. It corresponds to the type of system object which parameter values you want to set. Possible values: String type, range is limited by object types registered in the system.
2. **id** is a required argument. It corresponds to the identification (registration) number of the object specified by the **objtype** argument. Possible values: String type, range is limited by the identification numbers of objects of the specified type registered in the system.
3. **param<value>** is a required argument. You can specify several arguments of this type. It corresponds to the parameter(s) of the system object reaction.

Syntax for setting a value to a parameter corresponds to a string:

“param<value>”, where

param is a name of the parameter;

value is a value of the parameter.

Syntax for setting a value to several parameters corresponds to a string:

“param1<value1>,param2<value2>…”.

Elements of the list are separated by commas without spaces.

Possible values of the **param** argument: String type, range is limited by the available parameters of the specified reaction. Possible values of the **value** argument: String type, range depends on the parameter that you want to set.

Example. On macro 1 starts, temporary remove all cameras from the first monitor.

```
 if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
DoReactSetup ("MONITOR","1","REMOVE_ALL","");
}
```

The DoReactSetupCore method

The DoReactSetupCore method is used to change the parameters of the system object. The method changes only the specified parameters of the object, leaving other parameters unchanged.

Syntax for method invocation:

```
function DoReactSetupCore(objtype : String, id : String, param<value> [, param<value>] : String )
```

Method arguments:

1. **objtype** is a required argument. It corresponds to the type of system object which parameter values you want to set. Possible values: String type, range is limited by object types registered in the system.
2. **id** is a required argument. It corresponds to the identification (registration) number of the object specified by the **objtype** argument. Possible values: String type, range is limited by the identification numbers of objects of the specified type registered in the system.
3. **param<value>** is a required argument. You can specify several arguments of this type. It corresponds to the parameter(s) of the system object reaction.

Syntax for setting a value to a parameter corresponds to a string:

"param<value>", where

param is a name of the parameter;

value is a value of the parameter.

Syntax for setting a value to several parameters corresponds to a string:

"param1<value1>,param2<value2>…".

Elements of the list are separated by commas without spaces.

Possible values of the **param** argument: String type, range is limited by the available parameters of the specified reaction. Possible values of the **value** argument: String type, range depends on the parameter that you want to set.

Example. On macro 1, set the values of the following parameters of cameras 1-4: PTZ device number (telemetry_id) and number of the microphone for synchronous recording (audio_id). The values must be equal to the camera numbers plus 1.

```
 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) + ">");
 }
}
```

The DoReactGlobal method

The DoReactGlobal method is used to generate reactions of system objects. The DoReactGlobal method sends a reaction to the required object. The reaction is sent not only to the kernel on which the object is registered, but to the entire system. In the DoReactGlobal method, the reaction is specified by the **MsgObject** object.

Syntax for method invocation:

```
function DoReactGlobal(msgevent : MsgObject)
```

Method arguments:

1. **msgevent** is a required argument. It specifies the reaction sent to the specified object. Possible values: the **MsgObject** objects created earlier in the script.

Example. On macro 2, arm sensor 2. The command must be sent to all system kernels as a reaction to be registered in the Event Viewer.

```
 if (Event.SourceType == "MACRO"&& Event.SourceId == "2" && Event.Action == "RUN")
{
var msgevent = CreateMsg();
msgevent.SourceType = "GRAY";
msgevent.SourceId = "2";
msgevent.Action = "ARM";
DoReactGlobal(msgevent);
}
```

The NotifyEvent methods

The NotifyEvent methods are used to generate system events. They differ in the event distribution level and method signature:

Method	Sent to	Signature
NotifyEvent	Directly to the kernel on which the object is registered	Event is specified by the **MsgObject** object
NotifyEventStr	Directly to the kernel on which the object is registered	Event is specified by a group of String arguments
NotifyEventGlobal	All system kernels	Event is specified by the **MsgObject** object
NotifyEventGlobalStr	All system kernels	Event is specified by a group of String arguments

The NotifyEvent method

The NotifyEvent method generates system events. The generated event is sent directly to the kernel on which the object is registered, and not to the entire system. In the NotifyEvent method, the event is specified by the **MsgObject** object (see The MsgObject and Event objects and their built-in methods and properties).

Syntax for method invocation:

```
function NotifyEvent(msgevent : MsgObject)
```

Method arguments:

1. **msgevent** is a required argument. It specifies the event sent to the system. Possible values: the **MsgObject** objects created earlier in the script.

> **Note**
>
> Two types of system messages are available in *Axxon PSIM*: events and reactions. The events usually contain some information and are used as notifications sent to all *Axxon PSIM* kernels connected to each other during the architecture confi guration. The reactions are commands sent to specific objects. The reactions are transmitted only to the kernel on which the object is registered, and not to the entire system. The DoReactStr and DoReact methods are used to generate reactions. The NotifyEventStr and NotifyEvent methods are used to generate events.

Example. When the **Backup Archive** 1 module starts archiving video recordings, the analog output 1 of the Video Capture Device 2 is disabled. Send the command as an event to be registered in the Event Viewer.

> **Note**
>
> When running this script, the analog output 1 of the Video Capture Device 2 is not disabled.

```
 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);
}
```

The NotifyEventGlobal method

The NotifyEventGlobal method is used to generate system events. The generated event is sent to all system kernels connected via the network. In the NotifyEventGlobal method, the event is specified by the **MsgObject** object (see The MsgObject and Event objects and their built-in methods and properties)

Syntax for method invocation:

```
function NotifyEventGlobal (msgevent : MsgObject)
```

Method arguments:

1. **msgevent** is a required argument. It specifies the event sent to the system. Possible values: the **MsgObject** objects created earlier in the script.

Example. On macro 1, the first camera is set to record. The command must be sent to all system kernels as the event to be registered in the Event Viewer.

> **Note**
>
> When running this script, camera 1 is not set to record.

```
 if (Event.SourceType == "MACRO"&& Event.SourceId == "1" && Event.Action == "RUN")
{
var msgevent = CreateMsg();
msgevent.SourceType = "CAM";
msgevent.SourceId = "1";
msgevent.Action = "REC";
NotifyEventGlobal(msgevent);
}
```

The NotifyEventStr method

The NotifyEventStr method generates system events. The generated event is sent directly to the kernel on which the object is registered, and not to the entire system. In the NotifyEventStr method, an event is specified by a group of String arguments.

Syntax for method invocation:

```
function NotifyEventStr(objtype : String, id : String, event : String, param<value> [, param<value>] : String )
```

Method arguments:

1. **objtype** is a required argument. It corresponds to the type of system object for which you want to generate an event. Possible values: String type, range is limited by object types registered in the system.
2. **id** is a required argument. It corresponds to the identification (registration) number of the object specified by the **objtype** argument. Possible values: String type, range is limited by the identification numbers of objects of the specified type registered in the system.
3. **event** is a required argument. It specifies the event that you want to generate. Possible values: String type, range is limited by events available for the object of the specified type.
4. **param<value>** is a required argument. You can specify several arguments of this type. It corresponds to the parameter(s) of the system event.

Syntax for setting a value to a parameter corresponds to a string:

"param<value>", where

param is a name of the parameter;

value is a value of the parameter.

Syntax for setting a value to several parameters corresponds to a string:

"param1<value1>,param2<value2>…".

Elements of the list are separated by commas without spaces. If no parameter needs to be specified, an empty string is used:

```
DoReactStr("CAM","1","MD_START","");
```

Possible values of the **param** argument: String type, range is limited by the available parameters of the specified event. Possible values of the **value** argument: String type, range depends on the parameter that you want to set.

> **Note**
>
> Two types of system messages are available in *Axxon PSIM*: events and reactions. The events usually contain some information and are used as notifications sent to all *Axxon PSIM* kernels connected to each other during the architecture configuration. The reactions are commands sent to specific objects. A reaction is transmitted only to the kernel on which the required object is registered, and not to the entire system. The DoReactStr and DoReact methods are used to generate reactions. The NotifyEventStr and NotifyEvent methods are used to generate events.

Example. When an alarm is received, send the "panic lock" event for the corresponding region of the camera. For camera identification numbers are from 1 to 4, use region 1. For camera numbers from 5 to 10, use region 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", "");
}
```

The NotifyEventGlobalStr method

The NotifyEventGlobalStr method is used to generate system events. The generated event is sent to all system kernels connected via the network. In the NotifyEventGlobalStr method, the event is specified by a group of String arguments.

Syntax for method invocation:

```
function  NotifyEventGlobalStr(objtype : String, id  : String, event : String, param<value>  [, param<value>] :
String )
```

Method arguments:

1. **objtype** is a required argument. It corresponds to the type of system object for which you want to generate an event. Possible values: String type, range is limited by object types registered in the system.
2. **id** is a required argument. It corresponds to the identification (registration) number of the object specified by the **objtype** argument. Possible values: String type, range is limited by the identification numbers of objects of the specified type registered in the system.
3. **event** is a required argument. It specifies the event that you want to generate. Possible values: String type, range is limited by events available for the object of the specified type.
4. **param<value>** is a required argument. You can specify several arguments of this type. It corresponds to the parameter(s) of the system event.

Syntax for setting a value to a parameter corresponds to a string:

"param<value>", where

param is a name of the parameter;

value is a value of the parameter.

Syntax for setting a value to several parameters corresponds to a string:

"param1<value1>,param2<value2>…".

Elements of the list are separated by commas without spaces. If no parameter needs to be specified, an empty string is used:

```
NotifyEventGlobalStr("CAM","1","MD_START","");
```

Possible values of the **param** argument: String type, range is limited by the available parameters of the specified event. Possible values of the **value** argument: String type, range depends on the parameter that you want to set.

> **Note**
>
> Two types of system messages are available in *Axxon PSIM*: events and reactions. The events usually contain some information and are used as notifications sent to all *Axxon PSIM* kernels connected to each other during the architecture configuration. The reactions are commands sent to specific objects. A reaction is transmitted only to the kernel on which the required object is registered, and not to the entire system. The DoReactStr and DoReact methods are used to generate reactions. The NotifyEventStr and NotifyEvent methods are used to generate events.

Example. When an alarm is received, send the "panic lock" event for the corresponding region of the camera. For camera identification numbers are from 1 to 4, use region 1. For camera numbers from 5 to 10, use region 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", "");
}
```

The SetObjectParam method

The SetObjectParam method is used to set the values to system object parameters.

Syntax for method invocation:

```
function SetObjectParam(objtype: String, id: String, param : String, value : String)
```

Method arguments:

1. **objtype** is a required argument. It corresponds to the type of system object which parameter values you want to set. Possible values: String type, range is limited by object types registered in the system.
2. **id** is a required argument. It corresponds to the identification (registration) number of the object specified by the **objtype** argu ment. Possible values: String type, the range is limited by the identification numbers of objects of the specified type registered in the system.
3. **param** is a required argument. It corresponds to the parameter of the system object. Possible values: String type, range is limited by the parameters available for the given object.
4. **value** is a required argument. It corresponds to the value set to the param parameter of the system object. Possible values: String type, range depends on the set parameter.

Example. On Macro 1, check if cameras 1 to 4 are set to transmit color video. If a camera is set for black-and-white video transmission, switch it to the color mode (set the **true** ("1") value to the **Color** ("color") parameter).

```
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");
 }
 }
}
```

Note

If the object is active when the script is started (that is the settings panel of this object is open), then object parameters can not be changed by the SetObjectParam method. For example, if the settings panel for the **Camera 1** object is open and the script above is started, the operation mode of Camera 1 will not be changed for the color one.

The SetObjectState method

The SetObjectState method changes the state of objects.

Syntax for method invocation:

```
function SetObjectState(objtype : String, id : String, state : String)
```

Method arguments:

1. **objtype** is a required argument. It corresponds to the type of system object, the state of which you want to change. Possible values: String type, range is limited by object types registered in the system.
2. **id** is a required argument. It corresponds to the identification (registration) number of the object specified by the **objtype** argument. Possible values: String type, the range is limited by the identification numbers of objects of the specified type registered in the system.
3. **state** is a required argument. It corresponds to the state to which you want to switch the object. Possible values: String type, range is limited by the states available for this object.

Example. Check if camera 1 is armed every hour. If camera 1 is disarmed, arm it.

> **Note**
>
> You must create the **Timer** object with identification number 1 beforehand. Set the **Minutes** parameter of the **Timer** object to 30. The timer will go off every hour, for example, 09:30, 10:30, 11:30, and so on.

```
if (Event.SourceType == "TIMER" && Event.SourceId == "1" && Event.Action == "TRIGGER")
{
 if (GetObjectState("CAM", "1") == "DISARMED")
 {
 SetObjectState("CAM", "1", "ARMED");
 }
}
```

The DebugLogString method

The DebugLogString method outputs the user messages into the debug windows of the *Editor-Debugger* utility.

Syntax for method invocation:

```
function DebugLogString(output : String)
```

Method arguments:

1. **output** is a required argument. It specifies the message string that you want to output in the debug window of the *Editor-Debugger* utility. Possible values: String type.

Example. When any event from any of the microphones is registered in the system, output it in the debug window.

```
if (Event.SourceType == "OLXA_LINE")
{
 var msgstr = Event.MsgToString();
 DebugLogString("Event from the microphone " + msgstr);
}
```

The Base64Decode method

The Base64Decode method is used to decode strings that are encoded by the Base64 scheme.

Syntax for method invocation:

```
function Base64Decode(data_in: String, WideChar: Boolean)
```

Method arguments:

1. **data_in** is a required argument. It specifies the Base64 string that you want to decode;
2. **WideChar** is a required argument. It specifies the encoding type. Possible values: 0 or 1. If the encoding type is Unicode, the argument value is 1, otherwise 0.

Example. On macro 1, decode the string specified in Base64. Output the decoding result in the debug window of the *Editor-Debugger* utility (the result is the "Axxon PSIM JScript" string).

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
    var str = Base64Decode("SW50ZWxsZWN0IEpTY3JpcHQ= ", 0);
    DebugLogString(str);
}
```

The Sleep method

The Sleep method pauses the execution of the script for a specified period of time.

Syntax for method invocation:

```
function Sleep(milliseconds : int)
```

Method arguments:

1. **milliseconds** is a required argument. It specifies the time for which you want to pause the execution of a scrip. It is set in milliseconds. Possible values: int type.

Example 1. On macro 1, playback the following audio files one by one with audio player 1: cam_alarm_1.wav, cam_alarm_2.wav, cam_alarm_3.wav from the …\Axxon PSIM\Wav\ folder. The delay between the playback of each audio file must be five seconds (5000 milliseconds).

```
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);
 }
}
```

Example 2. On macro 2, start the timer 1 that triggers every 10 seconds for a minute from the start of macro 2.

> **Note**
>
> To start this script, you must create the **Timer** object with ID 1 beforehand. Leave the default object parameters (**Any**). The **Timer 1** object can be disabled.

```
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", "");
}
```

The Itv_var method

The **Itv_var** method sets and returns the values of global variables.

Syntax for method invocation:

```
function Itv_var (globalvar : String) : String
```

1. **Globalvar** is a required argument. It specifies the name of the global variable. Possible values: String type that meets the requirements for valid names of String parameters of the Windows registry.

> **Note**
>
> Global variables are stored in the Windows registry to maintain their values after Windows restart. All global variables are stored in the registry branch HKEY_USERS\S-1-5-21-…\Software\VMSScript\VMSSCRIPT and HKEY_CURRENT_USER\Software\VMSScript\VMSSCRIPT. To access a global variable directly from the registry, search the registry for it by its name.

Example. On macro 1, save the value of the bright parameter of camera 10 to the cam10bright global variable. On macro 2, set the bright parameter of the cameras 1 to 4 to the value of the cam10bright global variable.

```
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);
  }
 }
}
```

The Int_var method

The **Int_var** method sets and returns values of global variables of integer type.

> **Attention!**
>
> The **Int_var** method uses the same storage as the Itv_var method, but modifies the type of variable to the integer type.

Syntax for method invocation:

```
function Int_var (globalvar : String) : int
```

1. **Globalvar** is a required argument. It specifies the name of the global variable. Possible values: String type that meets the requirements for valid names of String parameters of the Windows registry.

> **Note.**
>
> Global variables are stored in the system registry to maintain their values after Windows restart. All global variables are stored in the registry branch HKEY_USERS\S-1-5-21-…\Software\VMSScript\VMSSCRIPT and HKEY_CURRENT_USER\Software\VMSScript\VMSSCRIPT. To access a global variable directly from the registry, search the registry for it by its name.

Example. In the test example below, to check the method operation, the global variable named 2 is assigned a value of 1 that is then increases by 1 and is displayed in the debug window of the script.

```
if(Event.Action =="RUN")
{
  Int_var(2) = 1;
  Int_var("2")++;
  DebugLogString(Int_var("2").toString());
}
```

The GetIPAddress method

The GetIPAddress method returns the IP address of the *Axxon PSIM* kernel according to the existing architecture of the distributed video surveillance system.

Syntax for method invocation:

```
function GetIPAddress (dst : String, src : String) : String
```

Method arguments:

1. **dst** is a required argument. It specifies the name of the remote computer on which the *Axxon PSIM* kernel is installed. The value of the **dst** argument must match one of the computer names registered when configuring the architecture of the distributed video surveillance system. Possible values: String type that meets the requirements for network computer names; the range is limited to the computer names registered in the system.
2. **src** is a required argument. It specifies the name of the local computer (the computer from which you run the script). The value of the **src** argument must match the name of the local computer as it is registered in *Axxon PSIM*. Possible values: String type that meets the requirements for network computer names.

> **Note**
>
> The information about all connections of the local computer (kernel) to other remote computers (kernels) registered during the configuration of the distributed architecture is displayed in the **Architecture** tab of the **System settings** window.

Example. On a camera alarm, determine the name of the computer to which this camera is connected, and output in the debug window the IP address of the connection between this computer and the local computer on which the script is executed.

```
if (Event.SourceType == "CAM" && Event.Action == "MD_START")
{
 var camid = Event.SourceId;
 var compname = GetObjectParentId("CAM", camid, "COMPUTER");
 var ip = GetIPAddress("WS1","WS1"); \\if the script is run on the computer where kernel of Axxon PSIM software
has been installed
 DebugLogString("IP-address of the alarmed camera computer" + ip);
}
```

> **Note**
>
> Instead of "WS1" in the example, you must enter the name of the computer on which the script is run and *Axxon PSIM* kernel is installed.

The CreateMsg method

The CreateMsg method creates objects based on the **MsgObject** prototype (see The MsgObject and Event objects).

Syntax for method invocation:

```
function CreateMsg() : MsgObject
```

There are no method arguments.

Example 1. When an alarm is received, send the "panic lock" event for the corresponding region of the camera. If the identification number of the alarm camera is from 1 to 4, use region 1. If the identification number of the alarm camera is from 5 to 10, use region 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);
}
```

Example 2. When timer №1 starts, start macro 1 every 30 seconds.

> **Note**
>
> To start this script, you must create the **Timer** object with the identification number 1 beforehand. Set value 1 to the **Second** parameter of the **Timer** object, leave other parameters unchanged (**Any** by default).

```
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", "");
 }
}
```

The Lock and Unlock methods

The Lock and Unlock methods are used to create a global critical section when synchronization of scripts started in different streams is required. The Lock method opens a critical section and the Unlock method closes it.

> **Attention!**
>
> If you invoke the Lock method, you must also invoke the Unlock method. Otherwise, the system can freeze.

We recommend avoid using the Lock and Unlock methods.

Syntax for method invocation:

```
function Lock()
```

```
function Unlock()
```

Example. On macro 1, calculate the total amount of alarmed relays and sensors. Objects of each type must be calculated at the same time (in an individual script). The result must be written to the counter global variable.

Script 1:

```
 // Number of alarmed relays is calculated
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();
}
}
```

Script 2:

```
 //Number of alarmed sensors is calculated
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();
}
}
```

Note

If the Lock() and Unlock() methods are not used in this example, then collisions can occur and the calculated value will be less than the actual value.

The IsAvailableObject method

The IsAvailableObject method is used to determine the current access permissions to an object.

Syntax for method invocation:

```
function IsAvailableObject(compname: String, objtype: String, id: String, param : String) : String
```

The method returns 0 if the current user has not been assigned permissions of the **param** type to access the object. The method returns 1 if permissions have been assigned.

Method arguments:

1. **compname** is a required argument. It corresponds to the name of the **Computer** object on the basis of which the object was created in the hardware tree.
2. **objtype** is a required argument. It corresponds to the type of the system object access permissions to which you want to find out. Possible values: String type, range is limited by object types registered in the system.
3. **id** is a required argument. It corresponds to the identification (registration) number of the object specified by the **objtype** argument. Possible values: String type, range is limited by the identification numbers of objects of the specified type registered in the system.
4. **param** is a required argument. It corresponds to the number of the type of permissions that you want to find out if they exist. A description of permissions is given in Limiting access to the system objects administration, control and viewing functions. Possible values:
    a. 0—rightsNoView access permissions. The method will return 1 if there are no administrative, control, or monitoring permissions to the object (red "x").
    b. 1—rightsNoControl access permissions. The method will return 1 if there are only monitoring permissions (letter M).
    c. 2—rightsViewAndControl access permissions. The method will return 1 if there are control and monitoring permissions to the object (green checkbox).
    d. 3—rightsViewOrControl access permissions. The method will return 1 if there is either monitoring or control permissions to the object.
    e. 4—rightsNot access permissions.
    f. 5—rightsConfigure access permissions. The method will return 1 if there are administrative, control, and monitoring permissions to the object (grey checkbox).

Example. A **Camera** object with the identifier 1 has been created in the hardware tree on the basis of the **Computer** object with the "Comp" name. Find out the current access permissions to the object.

```
var i = 0;
for(i = 0; i <= 5; i++)
{
     var result =
    IsAvailableObject('Comp','CAM','1', i);
    DebugLogString("right "+i+" = "+result);
}
```

The GetUserId method

The GetUserId method returns the identifier of the current *Axxon PSIM* user.

Syntax for method invocation:

```
function GetUserId (cmp : String) : String
```

Method arguments:

1. **cmp** is a required argument. It specifies the name of the computer on which *Axxon PSIM* is installed. Possible values: String type that meets requirements for network computer names, range is limited by computer names registered in the system.

Example. Display in the debug window the identifier of the current user of *Axxon PSIM* installed on a computer named 'WS3':

```
DebugLogString(GetUserId("WS3"));
```

The GetEventDescription method

The GetEventDescription method is used to get a description of an event in natural language.

Syntax for method invocation:

```
function  GetEventDescription (obj_type : String, event : String)
```

Method arguments:

1. **obj_type** is a required argument. It specifies the type of system object, the description of which you want to get.
2. **event** is a required argument. It specifies the name of the event, the name of which you want to get.

Example. When receiving events from camera 1, display messages about them in natural language in the debug window.

```
if (Event.SourceType == "CAM"&& Event.SourceId == "1")
{
        var str = GetEventDescription("CAM", Event.Action);
        DebugLogString(str);
}
```

The SaveToFile method

The SaveToFile method is used to save to a file the frame from the camera that is received in the data parameter of the FRAME_SENT event.

Syntax for method invocation:

```
function SaveToFile (path: String, data: String, param : Boolean)
```

You can also save the frame using the GET_FRAME reaction of the CAM object. To do this, you must specify the path for saving the file with the frame in the path parameter of this reaction. The FRAME_SENT event is created in the system if the GET_FRAME reaction doesn't have the path parameter. In the data parameter of the FRAME_SENT event, the frame that must be saved using the SaveToFile method is stored.

The reaction allows you to export a frame even if the camera is not displaying in the Video surveillance monitor.

Method argument:

1. **path** is a required argument. It specifies the full path to save the file with a frame.
2. **data** is a required argument. It specifies data to save to a file.
3. **param** is a required argument. It determines whether to transcode from base64 format before saving. Possible parameter values:
    a. true—transcode from base64 before saving;
    b. false—save the string without transcoding.

Time of frame saving depends on the reference frame rate. The higher the reference frame rate, the shorter the time.

Example. When a frame is received from Camera 1, save it to the test.jpg file on disk D:

```
if (Event.SourceType == "CAM" && Event.SourceId == "1" && Event.Action == "FRAME_SENT")
 {
 SaveToFile("D:\\test.jpg",Event.GetParam("data"),true);
 }
```

The GetLinkedObjects method

The GetLinkedObjects method is used to get the list of objects linked to the specified camera using the **Objects link** object (see Connection of objects with cameras).

Syntax for method invocation:

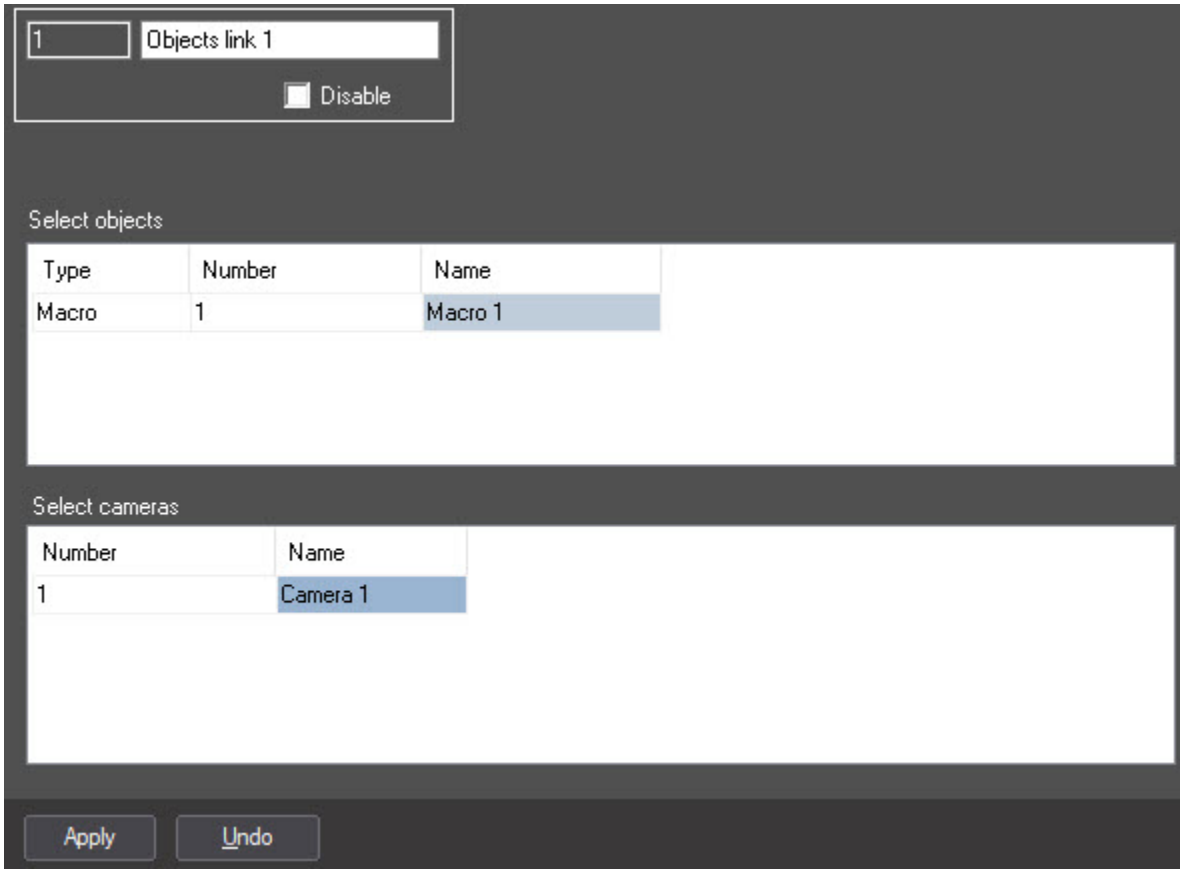
```
function GetLinkedObjects(type1 : string, id : string, type2 : string)
```

Method argument:

1. **type1** is the type of object for which you want to return the linked objects.
2. **id** is the identification number of an object for which you want to return the linked objects.
3. **type2** is the type of linked objects which you want to return. If empty string is sent, linked objects of all types will be returned.

Example.

The **Objects link** object is configured in the following way:



Display in the debug window the list of objects linked to camera 1.

```
if (Event.SourceType == "MACRO")
{

varmsgstr = GetLinkedObjects("CAM","1","MACRO")

DebugLogString("Linked objects " + msgstr);

}
```

As a result, the "Linked objects MACRO:1" message will be displayed in the script debug window.

The WriteIni method

The WriteIni method is used to write the string variable to the ini file.

Syntax for method invocation:

```
function WriteIni(varName: String, varValue: String, path: String)
```

Method argument:

1. **varName** is a required argument. It specifies the name of the variable for storing in the file.
2. **varValue** is a required argument. It specifies the value of the variable.
3. **path** is a required argument. It specifies the full path to the ini file in which the variable must be stored. Storage of variables can be placed on the network resource. To do this, enter the network path in the argument.

Example. Write the MyVar variable to the \\fileserver\temp\test.ini file and set the "Hello world!" value to it. Then read the written value and display it in the script debug window.

```
WriteIni("MyVar", "Hello world", "\\\\fileserver\\temp\\test.ini");
var result = ReadIni("MyVar","\\\\fileserver\\temp\\test.ini");
DebugLogString(result);
```

The ReadIni method

The ReadIni method is used to read values of the string variable in the ini file.

Syntax for method invocation:

```
function ReadIni (varName: String, path: String): String
```

Method arguments:

1. **varName** is a required argument. It specifies the name of the variable stored in the file.
2. **path** is a required argument. It specifies the full path to the ini file in which the variable is stored.

See example in The WriteIni method.

The AddIni method

The AddIni method is used to write, change and read integer variable from the ini file. The method returns the value of the variable after its changing.

Syntax for method invocation:

```
function AddIni(varName: String, varValue: int, path: String): int
```

1. **varName** is a required argument. It specifies the name of the variable in the file.
2. **varValue** is a required argument. It specifies the value of the variable or the value which must be added to the existing value of the variable:

    a. If a variable with the varName name and a string value is stored in the file, the varValue value will be assigned to the variable.
    b. If there is no variable with the varName name in the file, such a variable will be created and the varValue value will be assigned to it.
    c. If there is a variable with the varName name in the file and it has an integer value or its value can be brought to the integer type, the value will be brought to the integer type and the varValue value will be added to it.
3. **path** is a required argument. It specifies the full path to the ini file in which the variable must be stored. Storage of variables can be placed on the network resource. To do this, enter the network path in the argument.

Example. There is no the MyVar variable in the C:\\test.ini file. Write to this file such a variable with the -1 value, then add 1 to it and display the result value in the script debug window.

```
var result = AddIni("MyVar", -1, "C:\\test.ini");

result = AddIni("MyVar", 1, "C:\\test.ini");

DebugLogString(result);
```

The SetTimer method

The SetTimer method is used to start the timer.

Syntax for method invocation:

```
function SetTimer (id : int, milliseconds : int)
```

Method arguments:

1. **id** is a required argument. It specifies the timer ID. Possible values are int or string type.
2. **milliseconds** is a required argument. It specifies the period with which the timer will trigger if it is not stopped by the KillTimer method. It is specified in milliseconds. Possible values: int type.

Example. Two seconds after the macro 1 is executed, start recording on camera 1.

```
if(Event.SourceType=="LOCAL_TIMER" && Event.Action=="TRIGGERED" && Event.SourceId==333) //you can specify Event.
SourceId == "333", that is use the string identifier type
{
 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
}
```

The KillTimer method

The KillTimer method is used to stop the timer. It returns 1 if the timer was stopped as a result of the function execution.

Syntax for method invocation:

```
function KillTimer (id : int) : int
```

Method arguments:

1. **id** is a required argument. It specifies the timer ID. Possible values: int or string type.

Example. See in The SetTimer method.

The Base64EncodeFile method

The Base64EncodeFile method is used to encode files using the Base64 scheme. The method returns a string.

See also The Base64Decode method and The SaveToFile method.

Syntax for method invocation:

```
function Base64EncodeFile (data_in: String): String
```

Method arguments:

1. **data_in** is a required argument. It specifies the path to the file that you want to encode.

Example. On macro 1, encode the 1.bmp file to Base64 and write it to the 2.bmp file.

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
        var s = Base64EncodeFile("d:\\1.bmp");
        SaveToFile("d:\\2.bmp",s, true);
}
```

The Base64EncodeW method

The Base64EncodeW method is used to encode a Unicode string using the Base64 scheme. The method returns a string.

See also The Base64Decode method.

Syntax for method invocation:

```
function Base64EncodeW (data_in: String): String
```

Method arguments:

1. **data_in** is a required argument. It specifies the string that you want to encode.

Example. Decode a string from Unicode to Base64, encode it back and display it in the debug string.

```
var test = Base64Decode
("MAAzAC0AMAA3AC0AMgAwADEAOQAgADEANQA6ADMAOQA6ADQAMAA6AA0ACgB0AGUAcwB0ACAAMQANAAoAMAAzAC0AMAA3AC0AMgAwADEAOQAgAD
EANQA6ADQAMgA6ADIAMQA6AA0ACgB0AGUAcwB0ACAAMgA=",true);
DebugLogString("----->"+test);
var res = Base64EncodeW(test);
DebugLogString("----->"+res);
```

If the Base64Decode method received the true parameter, the script will have the following output:

```
03-07-2019 15:39:40:
test 1
03-07-2019 15:42:21:
test 2
```

The run_cmd and run_cmd_timeout methods

The run_cmd method is used to execute commands on the command line from a script.

The run_cmd_timeout method is used to execute commands on the command line with the specified process termination timeout.

When invoking commands, the command line window does not open, the commands are executed in hidden mode.

Syntax for method invocation:

```
function run_cmd (cmd: String)
function run_cmd_timeout (cmd: String, timeout: int)
```

Method arguments:

1. **cmd** is a command for the command line.
2. **timeout** (only for run_cmd_timeout) is a command line process termination timeout after the command execution.

Example 1. Run the curl utility and send a POST request with the text "Hello" to the test URL https://postman-echo.com/post

```
var s = run_cmd("curl --request POST --url https://postman-echo.com/post --data \'Hello\'");
DebugLogString(s);
```

Example 2. Display the CPU usage on Chart 1, updating information on Timer 3.

```
var id = "1"; // id of the Charts object
var timer_id = "3"; // id of the Timer object to trigger the script
SLAVE_id = "DESKTOP-5397BVV"; // id of the Computer object

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+">");
}
```

The WriteIniAny method

The WriteIniAny method is used to write a string variable to an ini file. Unlike the WriteIni method, in the WriteIniAny method you can specify the required file section for writing.

Syntax for method invocation:

```
function WriteIniAny(varName: String, varValue: String, path: String, section: String)
```

Method arguments:

1. **varName** is a required argument. It specifies the name of the variable for storing in the file.
2. **varValue** is a required argument. It specifies the value of the variable.
3. **path** is a required argument. It specifies the full path to the ini file in which the variable must be stored. Storage of variables can be placed on the network resource. To do this, enter the network path in the argument..
4. **section** is a required argument. It specifies the name of the section of the ini file in which you want to write the variable.

Example. Write the MyVar variable to the "config" section of the C:\\Backup\\test2.ini file, and set the "Hello world!" value to it. Then read the written value and display it in the debug window of the script.

```
WriteIniAny("MyVar", "Hello world!", "C:\\Backup\\test2.ini", "config");
var result = ReadIniAny("MyVar","C:\\Backup\\test2.ini", "config");
DebugLogString(result);
```

The ReadIniAny method

The ReadIniAny method is used to read the values of a string variable in the ini file. Unlike the ReadIni method, in the ReadIniAny method you can specify the required file section from which you want to read the variable.

Syntax for method invocation:

```
function ReadIniAny (varName: String, path: String, section: String): String
```

Method arguments:

1. **varName** is a required argument. It specifies the name of the variable stored in the file.
2. **path** is a required argument. It specifies the full path to the ini file in which the variable is stored.
3. **section** is a required argument. It specifies the name of the section of the ini file from which you want to read the variable.

See example in The WriteIniAny method.

The AddIniAny method

The AddIniAny method is used to write, change and read integer variable from the ini file. Unlike the AddIni method, in the AddIniAny method you can specify the section of the file that contains the integer variable. The method returns the value of the variable after its changing.

Syntax for method invocation:

```
function AddIniAny(varName: String, varValue: int, path: String, section: String): int
```

1. **varName** is a required argument. It specifies the name of the variable in the file.
2. **varValue** is a required argument. It specifies the value of the variable or a value which must be added to the existing value of variable:
    a. The varValue value will be assigned to the variable if there is a variable with the varName name and a string value in the file.
    b. If there is no variable with the varName name in the file, such a variable will be created and the varValue value will be assigned to it.
    c. If there is a variable with the varName name in the file and it has an integer value or its value can be brought to the integer type, the value will be brought to the integer type and the varValue value will be added to it..
3. **path** is a required argument. It specifies the full path to the ini file in which the variable must be stored. Storage of variables can be placed on the network resource. To do this, enter the network path in the argument.
4. **section** is a required argument. It specifies the name of the section of the ini file in which the variable is stored.

Example. There is no MyVar variable in the config section of the C:\\test.ini file. Write to this file such a variable with the -1 value, then add 1 to it and display the result value in the script debug window.

```
var result = AddIniAny("MyVar", -1, "C:\\test.ini", "config");

result = AddIniAny("MyVar", 1, "C:\\test.ini", "config");

DebugLogString(result);
```

The MsgObject and Event objects and their built-in methods and properties

The MsgObject and Event objects

The **MsgObject** object is a prototype (template) used to create objects. It implements methods and properties used to process system events in *Axxon PSIM*. The methods and properties of the **MsgObject** object allow you to get information about system objects that send or receive events, generate reactions for system objects, change their states, and so on.

You can invoke the methods and properties of the **MsgObject** object-prototype via the objects declared and initialized on its basis, or via the **Event** static object.

The **Event** object is a static object that implements the interface for invoking the system events of *Axxon PSIM*. The **Event** object provides access to the system event that started the script. All methods and properties of the **MsgObject** prototype are available when working with the **Event** object.

You can declare (create) objects on the basis of the **MsgObject** prototype using the CreateMsg method of the **Core** object (see The CreateMsg method).

The GetSourceType method

The GetSourceType method returns the system type of the **MsgObject** or **Event** object.

Syntax for method invocation:

```
function GetSourceType() : String
```

There are no method arguments.

Example. On macro 1, arm Detection Zones *.1 in the **Day** mode for cameras 1–4. On macro 2, arm Detection Zones *.2 in the **Night** mode for cameras 1–4. On macro 3, arm Detection Zones *.3 in the **Rain** mode for cameras 1–4.

> **Note**
>
> The * charachter corresponds to identification number of a camera in the system (from 1 to 4).

```
 if(Event.GetSourceType() == "MACRO" && Event.GetAction() == "RUN")
{
 var k;
 //Switching the cameras to the Day mode by arming the *.1 detection zones
 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", "");
 }
 }

 //Switching the cameras to the Nigh mode by arming the *.2 detection zones
 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", "");
 }
 }

 //Switching the cameras to the Rain mode by arming the *.3 detection zones
 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", "");
 }
 }
}
```

The GetSourceId method

The GetSourceId method returns the identification (registration) number of the **MsgObject** or **Event** object.

Syntax for method invocation:

```
function GetSourceId() : String
```

There are no method arguments.

Example. See the example in The GetSourceType method.

The GetAction method

The GetAction method returns the event received as an **Event** object or specified for a **MsgObject** object.

Syntax for method invocation:

```
function GetAction() : String
```

There are no method arguments.

Example. See the example in The GetSourceType method.

The GetParam method

The GetParam method returns the value of the specified parameter of the system object for the **MsgObject** or **Event** object.

Syntax for method invocation:

```
function GetParam(param: String) : String
```

Method arguments:

1. **param** is a required argument. It corresponds to the parameter name of the system object for which the **MsgObject** object (or the **Event** static object) was created. Possible values: String type, range is limited by the parameters available for the given object.

> **Note**
>
> If the object has no parameter with this name, then the method returns an empty string.

Example. When any event from any camera is registered, check from which computer the event comes. If the computer has the WS3 name, create the event copy in which the computer's name is "Computer".

```
 if (Event.SourceType == "CAM")
{
var msg = Event.Clone();
if (msg.GetParam("SLAVE_id") == "WS3")
{
msg.SetParam("SLAVE_id", "Computer");
NotifyEvent(msg);
}
}
```

The SetParam method

The SetParam method sets a value to the specified parameter of the **MsgObject** or **Event** object. The method changes only the specified parameters of an object, other parameters remain unchanged.

Syntax for method invocation:

```
function SetParam(param : String, value : String)
```

Method arguments:

1. **param** is a required argument. It corresponds to the parameter name of the system object for which the **MsgObject** object (or the **Event** static object) was created. Possible values: String type, range is limited by the parameters available for the given object.
2. **value** is a required argument. It sets the value to the parameter specified in the **param** argument. Possible values: String type, range depends on the set parameter.

Example. See the example in GetParam method.

The MsgToString method

The MsgToString method converts the **MsgObject** objects (including the **Event** static object) into a String variable.

Syntax for method invocation:

```
function MsgToString() : String
```

There are no method arguments.

Example. Send messages about all events registered for microphone 1 to a specified email address.

> **Note**
>
> The **Short Messages Service** must be installed and work properly.

```
 if (Event.SourceType == "OLXA_LINE" && Event.SourceId == "1")
{
 var msgstr = Event.MsgToString();
DoReactStr("MAIL_MESSAGE", "1", "SEND", "subject<Microphone 1>,body<" + msgstr + ">");
DoReactStr("MAIL_MESSAGE", "1", "SEND", "");
}
```

The StringToMsg method

The StringToMsg method converts a String variable into a **MsgObject** object.

Syntax for method invocation:

```
function StringToMsg(msg : String) : MsgObject
```

Method arguments:

1. **msg** is a required argument. It specifies a variable of the String type that you want to convert into a **MsgObject** object. Possible values: variables of the String type that match the syntax of the **MsgObject** objects representation:

"objtype|id|action|param1<value1>,param2<value2>…", where

- **objtype** is a type of system object;
- **id** is an identification number of a system object;
- **action** is an event or reaction of a system object;
- **param1<value1>,param2<value2>** is a list of parameters with their values. Elements of the list are separated by commas without spaces. If no parameters need to be specified, an empty string is used after the vertical line (|), for example: "CAM|1|MD_START|"

Example. On alarm from Sensor 1 and 3, start recording audio from Microphone 1. On alarm from Sensor 2 or 4, start recording audio from Microphone 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);
}
```

The StringToParams method

The StringToParams method converts a String variable into the list of parameters and overwrites the existing list of parameters of the **MsgObject** object.

Syntax for method invocation:

```
function StringToParams(params: String)
```

Method arguments:

1. **params** is a required argument. It specifies a variable of the String type that you want to convert into a list of parameters of the **MsgObject** object. Possible values: variables of the String type that match the syntax of the **MsgObject** objects parameter list representation:

“param1<value1>,param2<value2>…”, where

param1<value1>,param2<value2> is a list of parameters with their values. Elements of the list are separated by commas without spaces. If no parameters need to be specified, an empty string is used after the vertical line (|), for example: “CAM|1|MD_START|”

Example. When registering an **Attach** event for any of the cameras, re-initiate the **Attach** event in the system with changed values of the **Number of the PTZ device** (telemetry_id) and **Number of the microphone for synchronous recording** (audio_id) parameters. The values must be equal to the corresponding camera numbers plus 1.

```
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);
 }
}
```

The Clone method

The Clone method creates a copy of a **MsgObject** and **Event** object.

Syntax for method invocation:

```
function Clone() : MsgObject
```

There are no method arguments.

Example. When relay 1 closes, start video recording on camera 1 and close relay 2. When relay 1 opens, start video recording on camera 2 and open relay 2.

```
 if (Event.SourceType == "GRELE" && Event.SourceId == "1")
{
 var msgevent = Event.Clone();
 if(Event.Action == "ON")
 {
 msgevent.SourceId = "2";
 DoReact(msgevent);
 DoReactStr("CAM","1","REC","");
DoReactStr("GRELE","2","ON","");

 }
 if(Event.Action == "OFF")
 {
 msgevent.SourceId = "2";
 DoReact(msgevent);
 DoReactStr("CAM","2","REC","");
DoReactStr("GRELE","2","OFF","");

 }
}
```

The GetObjectIds method

GetObjectIds method is responsible for receiving identifiers from all objects of a specified type.

Syntax for method invocation:

```
function GetObjectIds(objectType : String)
```

Method arguments:

1. **objectType** is a required argument. It specifies the type of the system object for which you want to return the value of the specified parameter (**CAM**, **GRAY**, **GRABBER**, and so on). Possible values: String type, range is limited by object types registered in the system.

A string is returned:

CAM||COUNT|id.3<5>,id.count<4>,id.0<2>,id.1<3>,id.2<4>

where

- id.count<> is a number of objects IDs,
- id.[number]<> is an object ID.

Example. On Macro 1, all cameras must be armed.

```
 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", "");
}
}
```

The GetObjectParams method

GetObjectParams method is used to get the parameters of an object.

Syntax for method invocation:

```
function GetObjectParams(objectType : String, objectId : String)
```

Method arguments:

1. **objectType** is a required argument. It specifies the type of the system object (**CAM**, **GRAY**, **GRABBER**, and so on) for which you want to return the type of the parent object. Possible values: String type, range is limited by object types registered in the system.
2. **objectId** is an object's identifier. Possible values: String type.

Example. On macro 1, check the color control of camera 2. If camera 2 is a color one, set it to recording.

```
 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", "");
}
}
```

The SourceType property

The SourceType property allows you to return and set the system type for the **MsgObject** or **Event** object.

Syntax for property invocation:

```
SourceType : String
```

Example. When relay 1 closes (for example, the button connected to the relay is pressed), print the frames from cameras 1 and 2.

```
 if (Event.SourceType == "GRELE" && Event.SourceId == "1" && Event.Action == "ON")
{
 //activating the Camera 1 Surveillance window
 DoReactStr("MONITOR","1","ACTIVATE_CAM", "cam<1>");
 //printing the frame from Camera 1
 DoReactStr("MONITOR","1","KEY_PRESSED","key<PRINT>");
 //activating the Camera 2 Surveillance window
 DoReactStr("MONITOR","1","ACTIVATE_CAM", "cam<2>");
 //printing the frame from Camera 2
 DoReactStr("MONITOR","1","KEY_PRESSED","key<PRINT>");
}
```

The SourceId property

The SourceType property allows you to return and set the identification number for the **MsgObject** or **Event** objects.

Syntax for property invocation:

```
SourceId : String
```

See the example in The SourceType property.

The Action property

The Action property allows you to return and set a reaction or event for the **MsgObject** or **Event** objects.

Syntax for property invocation:

```
SourceId : String
```

See the example in The SourceType property.

Programming guide. Conclusion

For more information about *Axxon PSIM*, see:

1. Administrator's Guide: configuring system objects in the interface.
2. Operator's Guide: working with *Axxon PSIM.*
3. Installing and configuring security system components guide: installing and configuring hardware (cameras, alarm systems, access control systems, and so on).

If you have any difficulties and problems while working with this software product, you can contact us. Before contacting us, we kindly ask you to answer the following questions:

1. What is the problem?
2. When did the problem occur and what had happened before it occurred?
3. Under what conditions does the problem occur?

The more detailed and precise information you give us, the faster our experts will resolve your problem.

We are striving to improve the quality of our products, and hence welcome any suggestions and comments regarding the operation of our software and its documentation.