Programming Guide

Programming guide. Introduction

Purpose of INTELLECT™ software

INTELLECT™ software is designed for the deployment of industrial scalable, flexible (adjustable) integrated security systems, based on the digital video surveillance and audio monitoring systems.

INTELLECT™ software has the following basic features:

1. Integration of digital video surveillance and audio monitoring systems with the existing data systems, various security equipment, third-party software, using integrated open interfaces of the data exchange.
2. Compatibility with diverse security devices and data systems, in particular, with the fire and security alarm and access control systems, video surveillance cameras, data analysis systems and systems for recognition of objects (events) and identification by their images.
3. Single-source registration and processing of events, generation of notifications and controlling response in compliance with the flexibly modified logics.
4. Ultimately unlimited capabilities for scaling, solution — specific adjustments, re-distribution of resources with changes in the number or quality of tasks in monitoring guarded locations and operating various equipment.

Setting logical interactions between objects in Intellect

Intellect capabilities are based on logical **interactions between objects. General information on ways of setting logical interactions are represented in the table:**

Method of setting logical interaction	Description	Implementation	Example
Setting panels of system objects	Base configuration of interaction between system objects	Implemented using functionality of system objects – see Administartor's Guide	Configuring video displaying in the **Monitor** box
Macro	Configuration of simple interactions between objects if their functionality can not perform the required actions.	Implemented using the **Macro** object – see Administrator's Guide	Enabling actuator (relay) when sensor is closed
Program	Configuration of complex interactions between objects if functionality of the **Macro** object cannot perform the required actions.	Implemented using the **Program** object as the code in the embedded programming language – see this guide.	Reset PTZ cameras and snapshot every 15 minutes
Script		Implemented using the **Script** object as JScript code – see Programming Guide (JScript)	

Purpose and structure of the guide

Programming Guide is a reference and information handbook on programming in embedded Intellect software language, which is designed for system administrators, installation and configuration technicians, users with administrator rights to the digital video surveillance and audio monitoring systems, developed on the basis of INTELLECT™ software.

Programming in INTELLECT™ enables automatic system control by setting up complex logical interactions between objects.

The guide contains the information on:

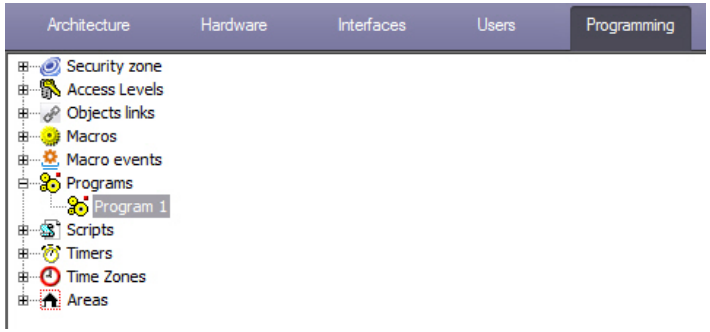
1. Programming tools;
2. Description of embedded programming language syntax;
3. Examples of programs in the embedded language.

Programming tools in Intellect

The Program system object

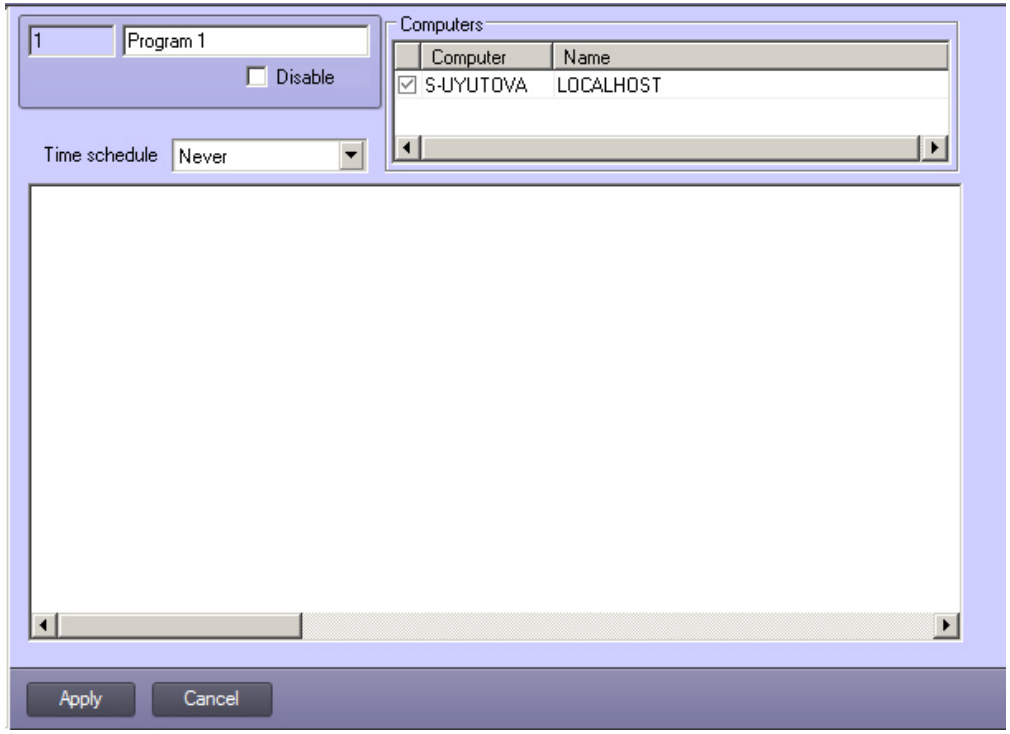
The **Program** system object is designed to initialize the program written in JScript (or Intellect programming language) in Intellect and set its parameters.

The **Program** system object is created under the **Programs** object in the **Programming** tab of the **System settings** dialog box.



> **Important!**
>
> 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 zone of program execution and computers (cores) on which the program is to be executed.

> **Note.**
>
> To set all checkboxes checked select one in the column and click Ctrl+A. To set all checkboxes unchecked select one and click Shift+A.

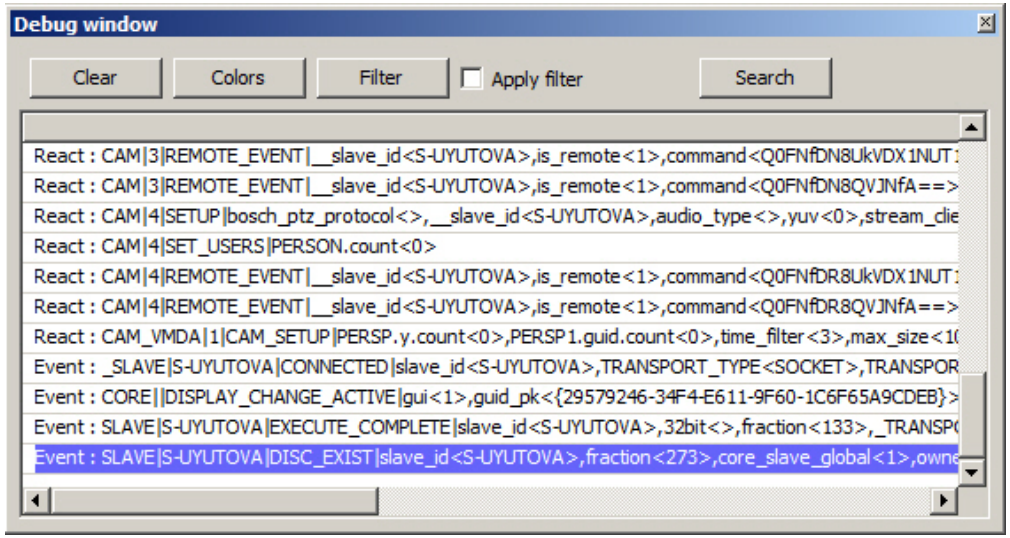
There is the word processor on the settings panel of the **Program** system object. It is used for writing and editing the program code.

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

Debug window

The Debug window is designed to view data about all events logged in the system.

Call the **Debug** window using the **Debug** command in the **Run** menu on the Main Control Panel. The Intellect **Debug** window appears at the bottom of the screen.



By default the **Debug window** is not available. Enable the **Debug window** using the *tweaki.exe* utility (see The Debug window section of Programming Guide (JScript)).

Syntax analyser

Embedded syntax analyser enables spell check of basic registered words, such as OnEvent, DoReact, OnTime, Wait, Sleep, etc. These registered words are black-marked in the text field. It is noteworthy that the analyser does not check if command parameters are written correctly, so you should 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+">");
```

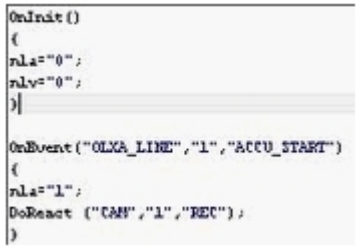
To change the font size use the key combinations

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

There should be a clear vision of what is to be done in the system as a result of specific events. Specify the ID of devices that participate in creating events and actions.

Outlining the task into subtasks

If several events are to 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, i.e. 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 loop 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 - from the server to 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, one may check and vice versa - make sure whether the event is written correctly events without starting the action list – run an empty macro and watch its performance in the debug window.

Finding and fixing bugs

At startup embedded syntax analyzer checks if names of functions are spelled correctly, but does not check the position of key characters - commas, semicolons and nested *parentheses* , etc. Therefore bugs, if any, will make themselves evident only at the stage of program body execution.

Description of syntax

Script consists of the set of procedures.

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

If a comment is to be written, then put // reserved characters before the comment.

Description of variables

All variables in the system are string variables.

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

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

Description of procedures

Standard procedures

There are 3 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. It is recommended to call the procedure once for all scripts.
   Example 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 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 of creating scripts. Examples 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 the Latin alphabet and the procedure will be executed for all events for which it can be defined.

Examples 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 are to be in the same program body and before procedures in which they are called.

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

Important!

The names of parameters are to consist of one uppercase character.

Examples 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 action
   Example 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 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 are to be in a single thread. Single thread is to be inside square brackets.
   Example**.** When Sensor 1 is closed, Relay 1 is closed for 5 seconds.

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

4. Checking the state function:
   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 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 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 cores, then use the NotifyEvent function.

Operators and expressions

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

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

+	Addition. Example 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. Example 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. Example 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. Example 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 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 use.
	`OnEvent ("MACRO","1","RUN")`
	`{`
	`  x=100/((5*8)/1.028);`
	`}`

Logical operators	

&&	Logical AND operator. Example 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 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 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 use**
MATH	
sin[1]	Trigonometric function: 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: 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 of an angle. 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 function value e^x, 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 of an angle. 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 of an angle. 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 since 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. (The 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 rubles and copecks.
	
	Format: x=convert_cur(y); where x- string value of the sum of money, y- number (sum of money).
	
	Recording format: <RR.KK>
	
	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 roubles ninty-eight copecks >,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 it 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]	Define 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]	Define 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 bigger than the number of symbols 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 *side* 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*>,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

On the page:

Below one can find some examples of scripts.

Example 1.

Display active camera on the analogue monitor.

Implementation:

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

Example 2.

Start and stop PTZ patrolling using macros.

Implementation:

```
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>");
}
```

Example 3.

Display an alarm camera in the single monitor mode.

Implementation:

```
OnEvent ("CAM",N,"MD_START")
{
    DoReact("MONITOR","1","ACTIVATE_CAM","cam<"+N+">");

    DoReact("MONITOR","1","KEY_PRESSED","key<SCREEN.1>");
}
```

Example 4.

Here is an example of an infinite loop and how to stop it. Macro1 starts the loop and macro2 ends it.

Implementation:

```
OnEvent("MACRO","1","RUN") //macro1 is run

{
    //square brackets are needed to isolate wait statement to individual stream
    [
    flag=1;
    for(a=1;flag<2;a=1) //cycle statement
    {
        Sleep(500); //wait statement creates pause of 500 milliseconds
        ff="!!!!!!!!!!!!!!!!!";
    }
    ]
}

OnEvent("MACRO","2","RUN") // macro2 is run
{
    flag=2;
}
```

Example 5.

An alarm monitor the video from the alarm camera is always on.

Implementation:

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

Example 6.

An audio file is to be played back starting from the moment when one event appears until the moment of another event appearance. (Macro is run in this case).

> **Important!!!**
>
> An audio file mustn’t be longer than the number of seconds specified in the Wait operator.

Implementation:

```
OnEvent("MACRO","1","RUN")
{
    flag=1;

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

}

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

Example 7.

There are 2 cameras with PTZ devices. Every 15 minutes a camera is to rotate to the position of preset 1 and a snapshot is to be made. Current time is the name for a file.

Implementation:

```
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);

    }

}
```

Example 8.

Audio from microphone (OLXA_LINE) is not recorded simultaneously with a camera. By default the microphone is not armed. Audio is to be recorded when being activated by sound and being detected by the camera.

> **Note.**
>
> RECORD_START and RECORD_STOP commands are added to the microphone in version 4.7.0 and later.

Audio recording is initiated by acoustic start (ACCU_START) and motion detection (MD_START) and variable flag is increased by 1. When acoustic start and motion detection end, then variable flag is decreased by 1 and audio recording is stopped only if it is 0, i.e. there is no acoustic start or motion.

Implementation:

```
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 9.

There are some cameras (num). Operation of motion detection on all cameras is to be checked (can also be used to check security sensors).

To solve this issue emulation of linear array of symbols (string) is in use, i.e. array of symbols is filled in ("N" here). When the motion detection is triggered on the camera, the corresponding element of the array (camera ID) changes (to "Y"). Thus, as a result we have array of "N" symbols (was not triggered) and of "Y" (was triggered).

The number of triggerings is counted and the message about total number of cameras and number of cameras on which the detection tool was triggered. The check is initiated by Macro1 and stopped by Macro2.

Implementation:

```
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="Total:"+str(num)+"Triggered:"+str(fin);
    rez=MessageBox("",tmp,0);

}
```

Example 10.

Patrol several zones using PTZ device presets; motion detection is to be enabled in some area of these zones.

Camera1. 5 detection zones and 5 presets. These parameters are set with *n* variable. Macro1 initiates algorithm execution. Macro2 ends algorithm execution. Flag - internal variable.

When the algorithm is started, the camera goes to preset1 and arms the 1st detection zone. The delay between these commands is 200 milliseconds (for the camera to go to preset). 5 seconds later the 1st zone is disarmed and the loop starts again but with the 2nd zone and preset2. And so on. Then it starts from the very beginning. The algorithm stops if variable flag resets to zero (using Macro2).

Implementation:

```
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 11.

There are 2 displays – the first displays a virtual monitor with cameras, the second displays the **Map** object with *Bolid* sensors. When alarm is triggered by the camera - Display 1 is shown, when alarm is triggered by the sensor - Display 2 is shown, but on the CLIENT only.

Implementation:

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

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

Example 12.

When an alarm event appears on camera 1, add subtitles to video from this camera. When the alarm event ends, add subtitles about the alarm end.

Implementation:

```
OnEvent("CAM","1","MD_START")
{
  DoReact("CAM","1","CLEAR_SUBTITLES","title_id<1>"); //clear all subtitles from the video
  DoReact("CAM","1","ADD_SUBTITLES","command<Camera 1 Alarm " + time + "\r>,page<BEGIN>,title_id<1>"); //time
parameter adds the time of event registering to subtitles
}
OnEvent("CAM","1","MD_STOP")
{
  DoReact("CAM","1","ADD_SUBTITLES","command<Camera 1 Alarm end " + time + "\r>,page<END>,title_id<1>");
}
```

Note.

When page<BEGIN> and page<END> parameters are in use, the corresponding fields of the subtitles database are filled in – this enables data search using the **Search by titles** interface object.

Description of system object reactions

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

Note.

It is possible to view events for system objects by one of the following ways:

1. Viewing the intellect.ddi file using the ddi.exe utility (see the Intellect Software Package. Administrator's Guide).
2. Viewing events for the selected object of system using the control panel of the **Macro** system object (see the Intellect Software Package. Administrator's Guide).

GRABBER

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 appears.

Format of events procedure for the video capture device:

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

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

Event	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 ventilator 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

Operator format to describe actions with the video capture device is:

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

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 – maximal 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<>	
	ip<>	IP-address of video input network 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 (change) parameters of IP-device. Reaction allows changing of IP-device settings not entering its web-interface. *Note. For reaction operation it is required to enable the mode of multi-flow video signal - see. Administrator's Guide, section Configuration of multistream video, and* Appendix 2. Defining param_id and param_value values for SET_IPINT_PARAM reaction	param_id<>	Name of parameter. Set of parameters for each camera is individual - see Appendix 2. Defining param_id and param_value values for SET_IPINT_PARAM reaction
	param_value<>	Value of parameter. Set of parameters for each camera is individual - see Appendix 2. Defining param_id and param_value values for SET_IPINT_PARAM reaction
	cam_id<>	Camera ID in the *Intellect* software package.
	vstream_id<>	Number of video-flow (optional parameter). Is given by "Number of camera"."Number of flow", for example 1.1, 1.2.

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.

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, maximal speed of digitizing, resolution is half-frame and PAL format while 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 maximal, resolution is half-
frame, format is PAL
}
```

> **Note.**
>
> Description of the MACRO object is follows (see the MACRO section).

2. Set disks D:\ and F:\ for video archive record while 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 video camera on the first analog output and disable first analog outputs of the first and second cards while 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 video 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 video camera 1 belongs to the first grabber and video camera2 belongs to the second grabber than while triggering the «DoReact("CAM","1","MUX1");» command it is required to trigger the «DoReact("GRABBER","2","MUX1_OFF");» command at first, and correspondingly while triggering the «DoReact("CAM","2","MUX1");» command it is required to trigger the «DoReact("GRABBER","1","MUX1_OFF");» command at first. Otherwise signal overlaying will happened.

Note.

Description of the **CAM** object is follows (see the CAM section).

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

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

CAM

The **CAM** object corresponds to the **Camera** system object.

The **CAM** object sends events given in the table. Procedure is started when the corresponding event appears.

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

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

Description of events of the **CAM** object.

Events	Description	Comment
"ARM"	Camera is armed.	

"ATTACH"	Connecting.	
"BLINDING"	Camera is sealed.	
"DETACH"	Break.	Event is generated while loss the input signal from camera on video capture device.
"DISARM"	Camera is disarmed.	
"FILE_REC_ERROR"	Error of recording on disk.	Event is generated while error of video archive recording on disk happens.
"MD_START"	Alarm.	
"MD_STOP"	End of alarm.	
"PRINT"	Print frame.	
"REC"	Recording on disk.	
"REC_STOP"	Stop recording on disk.	
"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 nonnull error_result parameter.
"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 via the Video Surveillance Monitor.

Operator format to describe actions with the camera is:

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

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) parameters of camera	rec_priority<>	Record priority (from 0 to 3, 0 – standard, 3 – all recources).
	compression<>	Compression ration (0 – no compression, 1- maximal quility, …, 5 – minimal quality).
	sat_u<>	Value of colour (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 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 (0 – 20 sec).
	mux<>	Number of channel (0 – 1 channel, 15 – 16 channel).
	color<>	Colour (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 anchor frames in sequence (1 – each frame is anchor, 2 – 100 frame).
	mask 0, mask1, mask2, mask3, mask4	Detection mask.
	md_contrast<>	Sensibility of motion detection (0 – 15).
	motion<>	Estimation of motion compressor (5 - 255).
	name<>	Name of object.
	password_crc<>	Password for video archive.
	priority<>	Priority of record resource (0 – auto, 1 – manual).
	resolution<>	Resolution (0 – standard CIF, 1 - high 2CIF, 2 –maximal 4CIF).
	type<>	Type of object.
	yuv<>	Colour schema of video signal coding (0 – YUV4:2:0, 1 - YUV4:2:2).
"DELETE" – disables camera.	-	-
"START_VIDEO" – enables video stream for current camera.	slave_id<>	Name of computer to which camera is connected.
	comress<>	Value of compression.
	register_only<>	-

"STOP_VIDEO" – disables video stream for current camera.	slave_id<>	Name of computer to which camera is connected.
"REQUEST_MASK"	mask<>	Mask.
"MUX1", "MUX2", "MUX3" – display image of camera on 1, 2, 3 analog outputs.	-	-
"ACTIVATE" – display camera on monitor.	monitor<>	Number of monitor.
"ARM" – arm camera.	-	-
"DISARM" – disarm camera.	-	-
"REC" – start record from camera.	time<>	Time of record in seconds, if null than only one frame is recorded.
	rollback<>	If one, than record is performed with rollback.
	priority<>	Set priority of command to start record. See Appendix 1. Priorities of start and stop recording commands
	stream_id<>	Set an identification number of stream for record. Stream ID is set as "n.m" where n is the Camera object ID, m is the number of the stream. *Note. If the specified stream is not in use 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 checked for it – see the The Settings panel of the Camera object section of the Administrator's Guide.*
"REC_STOP" – stop record from camera.	priority<>	Set priority of command to stop record. See Appendix 1. Priorities of start and stop recording commands
"SET_MASK" – set mask.	mask<>	Mask.
"ADD_SUBTITLES" – add titles.	command<>	Test of imposed titles.
	title_id<>	ID of **Captioner** object which is used to impose.
	page<>	Parameter allows record titles to the titles database to provide search by titles. Available values: BEGIN (start of record in database), END (end of record in database).
"SIP_CONNECT" - Sip connected	-	-
"SIP_DISCONNECT" – Sip disconnected	-	-

"SET_IPINT_PARAM" – Set (change) parameters of IP-device. Reaction allows changing of IP-device settings not entering its web-interface. *Note. For reaction operation it is required to enable the mode of multi-flow video signal - see. Administrator's Guide, section Configuration of multistream video, and* Appendix 2. Defining param_id and param_value values for SET_IPINT_PARAM reaction	param_id<>	Name of parameter. Set of parameters for each camera is individual - see Appendix 2. Defining param_id and param_value values for SET_IPINT_PARAM reaction
	param_value<>	Value of parameter. Set of parameters for each camera is individual - see Appendix 2. Defining param_id and param_value values for SET_IPINT_PARAM reaction
	vstream_id<>	Number of video flow (optional parameter). Is given by "Number of camera"."Number of flow", for example 1.1, 1.2.
GET_FRAME – get frame from camera even if it is not displayed in the Video surveillance monitor.	path<>	Path to save frame. If there is no parameter than the FRAME_SENT event with the data parameter will be formed in the system. Processing of this event is described in the section The SaveToFile method of the Programming Guide (JScript).
	stream<>	Optional parameter. Sets *Intellect* stream to get a frame from. The stream can be specified by number or purpose. Possible purposes: • stream_archive – stream for archive recording • stream_alarm – stream for archive recording at alarm • stream_client – stream for displaying • stream_analytic – stream for video analytics Stream number consists of camera ID and stream ID divided by dot, for example, 4.3 is for stream 3 from camera 4.
	time<>	Optional parameter. 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. Network name of the Video Gate to get the frame from.
	arch<>	Optional parameter. Network name of the Backup Archive to get the frame from.
	slave_id<>	Optional parameter. Network name of the Server to get the frame from.
	password_crc<>	Optional parameter. CRC sequence to be recorded to the file together with the frame.
ARCH_DEL_RECORD – delete archive recordings over the specified period.	fromTime<>	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.
	toTime<>	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.
	comment<>	Comment to a bookmark.
	slave_id<>	Computer and Video surveillance monitor IDs – the bookmark is created using them. Parameter format: <computer id >.<monitor id>. For example, slave_id<WS2.1> –WS2 is computer ID and 1 is Video surveillance monitor ID.
CRUISE_START – Auto cruise	cruise_id<>	Route name on camera
	action<>	Executed action: CRUISE_START – start cruising along the specified route. PATROL_PLAY – start patrolling along the specified route.
	cam_id<>	Camera ID
"GET_DEPTH" – get the archive depth. The ARCHIVE_DEPTH event from the SLAVE object (see SLAVE) 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 video gate archive. Example. `DoReactStr("CAM","1","GET_DEPTH","drive<V:\\>, gate");`

Properties of the **CAM** object are shown in the table.

Properties of the CAMobject	**Description of properties**
ID<>	Object ID
PARENT_ID<>	Parent object ID
TELEMETRY_ID<>	Telemetry module ID (ID of ptz)
REGION_ID<>	Region ID

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

1. Switch camera to the colored mode and start recording from it while arming the first camera.

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

2. Arm the first video camera while disabling the fifth video camera.

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

3. Use half of resources while recording from the first camera (i.e. if 4 video cameras are connected through the first video capture device than the first camera will record with speed 6 fps, and other three cameras – with speed 2-2,5 fps) if it is in alarm state.

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

4. Set maximal compression synchronously with the fourth microphone of audio card on the first video camera while recording from the first video camera on disk.

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

5. Start recording from the first camera with minimal quality in black and white mode when it stopped to be in alarm.

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

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

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

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

```
OnEvent("CAM","1","ATTACH") //first video 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>");
}
```

Check function of **CAM** object state:

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

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

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 appears.

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

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

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.
PLAY_START	-	Start the archive fragment playback
PLAY_STOP	-	Stop the archive fragment playback

Operator format to describe actions with the monitor is:

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

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 is to be removed from monitor.
"REMOVE_ALL" – removes all cameras from monitor.	-	-
"STOP_VIDEO" – stops video flow of camera.	cam<>	ID of camera in the settings tree video flow from which is to be stopped.
"REPLACE" – removes all cameras from monitor and triggers the specified camera.	slave_id<>	Name of computer to which monitor belongs, it is possible to place owner in script.
	cam<>	ID of camera in the settings tree which is to be displayed in the monitor.
	name<>	Name of camera which is to 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 (arming/disarming, record).
"ADD_SHOW" – adds cameras on the monitor.	cam<>	ID of camera in the settings tree which is to be displayed in the monitor.
	name<>	Object name which is to be displayed in the bottom-left corner.
	arch_id<>	-
	control<>	0 only archive viewing, 1 – it is also possible to control (arming/disarming, record).
"ACTIVATE_CAM" – activates camera.	cam<>	ID of camera in the settings tree which is to be activated.
"ARCH_FRAME_TIME" – search of video archive by date and time.	cam<>	-
	date<>	-
	time<>	-
	mode<>	Can take the following values:

		0 – Video Gate, if it is set (if not set, then the archive of the Video Server is searched) 1 – Video Server 2 – Long-term archive
"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 while 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.
"ACTIVATE" – activates control panel of monitor.	user_id<>	User ID.
	panel_active<>	-
"DEACTIVATE" – deactivates control panel of monitor.	-	-
"EXPORT_FRAME" – exports frame in JPG-file.	cam<>	-
	file	-
"KEY_PRESSED" – controls buttons of video surveillance monitor and video records archive.	number<>	-
	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 back of video surveillance windows; "CYCLE_FF" – scrolling straight of video surveillance windows; "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" – forward wind;

"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 function of auto scrolling of video surveillance windows frontwards. Period of scrolling is specified while configuring the Monitor interface object (see the Administrator's Guide, Configuring the display mode for camera boxes section). "STOP_CYCLE" – stop of auto scrolling of video surveillance windows.
"START_AVI_EXPORT" – starts video export *Note. See the example as follows.*	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" – start bookmarks export	-	-
"STOP_AVI_SCHEDULE" – stop bookmarks export	-	-
"CONTROL_TELEMETRY" – Telemetry control. See Mouse PTZ control section in Operator's Guide.	cam<>	ID of camera where mouse PTZ control is to be enabled or disabled.
	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.

Properties of the **MONITOR** object are shown in the table.

Properties of the MONITOR object	Description of properties
ID<>	Object ID.
PARENT_ID<>	Parent object ID.

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

1. Play record from video camera 1 on the monitor 4 with specified date and time while the first macro starting.

```
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 video camera of monitor 4 while 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 video camera is in alarm state and reset it when alarm will be 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. Enter the layout number one on the monitor screen while triggering 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 100) and using script:
   1. **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>
   2. **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>
   3. **Script (start by 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>");
}
```

PLAYER

The **PLAYER** object corresponds to the **Audio Player** system object.

Operator format to characterize actions with audio player is following:

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

List of commands and parameters for the **PLAYER** object is presented in the table.

Command – command description	**Parameters**	**Description**
"PLAY_WAV" – plays audio file.	file<>	Audio file with full path to it.
"SETUP" – settings of audio player parameters.	board<>	Sound unit of 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).

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.

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

1. Play audio file which address is «C:\ Program Files\Intellect\Wav\cam_alarm_1.wav» while enabling flag of audio player operation.

```
OnEvent("PLAYER","1","flags") // enable flag of audio player operation
{
    DoReact("PLAYER","1","PLAY_WAV","file< C:\ program files\intellect\wav\cam_alarm_1.wav >"); // play
audio file
}
```

OLXA_LINE

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 appears.

Format of event procedure for the microphone:

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

Event	Description
"ACCU_START"	Sound activated recording ON.
"ACCU_STOP"	Sound activated recording OFF.
"ARM"	Record ON.
"DISARM"	Record OFF.
"INCOMING_NUMBER"	Incoming telephone number.

"OUTGOING_NUMBER"	Outgoing telephone number.
"REC"	Start of record.
"REC_STOP"	End of record.
"RESET"	Microphone connecting.

Operator format to describe actions with the microphone is:

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

List of commands and parameters for the **OLXA_LINE** object is presented in the following table:

Command – command description	**Parameters**	**Description**
"ARM" – turn on the microphone to record.	-	-
"DISARM" – turn off the record from microphone.	-	-
"SETUP" – setting of microphone parameters.	type<>	Type of line.
	accu_start <>	Operating threshold of sound detection.
	accu_stop<>	Holding time of detection triggering.
	amp<>	Gain.
	aru<>	Automatic gain 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 record.

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.

Check function of the **OLXA_LINE** object state:

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

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 record.

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"); //disable record from microphone
}
```

2. Set minimal compression on microphone while disabling record of audio signal.

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

DIALOG

The **DIALOG** object corresponds to the **Operator query pane** system object:

Operator format to describe actions with the operator query pane is:

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

List of commands and parameters for the **DIALOG** object is presented in the following table:

Command – command description	**Parameters**	**Description**
"SETUP" – set up of operator query pane.	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 operator query pane.	-	-
"RUN_MODAL" – run operator query pane in modal mode.	-	-
"CLOSE" – close last opened operator query pane.	-	-
"CLOSE_ALL" – close all opened operator query panes.	-	-

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

1. Using macro 1 set coordinates of left top corner of the operator query pane (ptz video camera PANASONIC-850) in center of screen, forbid its moving and display it.

```
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 pane using macro 2.

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

MMS

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 appears.

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

Event	Description
"SET_CONNECTIONS"	List of available connections.

Format of operator to describe actions with mail message service is following:

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

List of commands and parameters for the MMS object is given in the table:

Command – command description	Parameters	Description
"SETUP" – settings for mail message service.	smtp<>	Address of SMTP server.
	connection<>	Type of connection.
	smtp_username<>	User name.
	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.

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

1. Set port number of the mail message server is 25 while triggering macro 1. OnEvent("MACRO","1","RUN")

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

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 appears.

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

Event	Description
“SEND_ERROR”	Error of message sending.
“SENT”	Message is sent.

Format of operator to describe actions with mail message is following:

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

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<>	Source address.
	to<>	Destination address.
	cc<>	Copies.
	subject<>	Message subject.
	body<>	Message body.
	attachments<>	Attachments. If several files are attached, their addresses are semicolon separated.
	flags<>	Flags.
	name<>	Object name.
	pack<>	Way of attachments packing.
"SEND" – send mail message.	-	-

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.

Example of using reactions of the **MAIL_MESSAGE** object.

1. Send message with image from video camera when it switches to alarm state while motion detection triggering.

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

OnEvent("CAM",N,"REC") //video 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 by camera>,
from<sergey.kozlov@itv.ru>, to<sergey.kozlov@itv.ru>,attachments<" + filename + ">");
    DoReact("MAIL_MESSAGE","1","SEND");
}
```

VMS

The **VMS** object corresponds to the **Voice Message Service** system object.

Operator format to describe actions with the voice message service is:

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

List of commands and parameters for the **VMS** object is presented in the following 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.

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

1. It is required to send message while performing macro 1 if modem is connected to COM2 port, type of dialing is tonal, do not wait for tonal signal.

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

GRELE

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 appears.

Format of events procedure for the relay:

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

Event	Description
"OFF"	Relay Off.
"ON"	Relay On.
"SIGNAL_LOST"	Connection lost.

Operator format to describe actions with the relay is:

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

List of commands and parameters for the **GRELE** object is presented in the following 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.

Check function of the **GRELE** object state:

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

The **GRELE** object can be in the following states:

State of the GRELE object	State description
"ON"	Relay ON.
"OFF"	Relay OFF.
"DETACHED_ON"	Connection lost.
"DETACHED_OFF"	Connection lost.

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

1. Enable relay 2 while connection with relay 1 is lost.

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

GRAY

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 appears.

Format of events procedure for the sensor:

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

Event	Description
"ALARM"	Alarm. This event is received while opening or closing the sensor (it depends on object settings) if sensor is armed. If sensor is disarmed then Sensor opened and Sensor closed events are received correspondingly.
"ARM"	Sensor is armed.
"CONFIRM"	Alarm received.
"DISARM"	Sensor is disarmed.
"NOT_VALID_STATE"	Zone is not ready.
"OFF"	Sensor opened. This event is received while sensor opening if sensor is disarmed.
"ON"	Sensor closed. This event is received while sensor closing if sensor is disarmed.
"SIGNAL_LOST"	Connection with sensor is lost

Operator format to describe actions with the sensor is:

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

List of commands and parameters for the **GRAY** object is presented in the following table:

Command – command description	**Parameters**	**Description**
"ARM" – arm sensor.	-	-
"DISARM" – disarm sensor.	-	-
"CONFIRM" – confirm alarm.	-	-
"SETUP" – settings for 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.

Check function of the **GRAY** object state:

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

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.
"DETACHED_DISARM"	Connection lost.
"OFF"	Normal.

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

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

```
OnEvent("GRAY","1"," SIGNAL_LOST") //connection with first sensor is losst
{
    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 video camera in case of 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 video camera
}
```

VNS

The **VNS** object corresponds to the **Voice notification service** system object.

Operator format to describe actions with the sensor is:

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

List of commands and parameters for the **VNS** object is presented in the following table:

Command – command description	**Parameters**	Description
"SETUP" – settings of the voice notification service.	card<>	Name of sound device. Note. Card name is to be correspond to name which is specified in settings of sound card of the **Voice notification service**.
	level<>	Level of signal. Value of parameter is from 0 to 15. On default it is 8.
	channel<>	Set of sound channels. Available values of 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<>	User name.
"PLAY" – play audio file.	file<>	Full path and name of sound file. Note. If only file name is specified then path to it will be taken from registry in «HKEY_LOCAL_MACHINE\SOFTWARE\ITV\Intellect» section (HKEY_LOCAL_MACHINE \Software\Wow6432Node\ITV\Intellect for 64-bits system), in value of the «InstallPath» parameter. In this parameter it is possible to play several audio files using the «+» operation.

Command – command description	**Parameters**	**Description**
"ARM" – arm sensor.	-	-
"DISARM" – disarm sensor.	-	-
"CONFIRM" – confirm alarm.	-	-
"SETUP" – settings for 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 **VNS** object are shown in the table.

Properties of the VNS object	Description of properties
ID<>	Object ID.
PARENT_ID<>	Parent object ID.

Examples of using events and reactions of the **Voice notification service** object:

1. It is required to play audio file when video camera starts record.

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

2. Set less level of the volume regulator in the event of the earlier specified time zone and set the average level of the volume regulator after this time zone.

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

Note.

The TIME_ZONE object is described as follows (see TIME_ZONE section).

SMS

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 appears.

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

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

Description of events of the **SMS** object.

Event	Description	Comment
RECEIVE	Message is received	Use the ProcessFromSim registry key if event is not received while message receiving (see Registry keys reference guide). Text of sent message is in the **message <>** parameter. The telephone number in the +7XXXXXXXXXX format from which message was sent is in the phone<> parameter.

Operator format to describe actions with the short message service is:

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

List of commands and parameters for the **SMS** object is presented in the following 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<>	Telephone 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.

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

1. It is required to send short message to the "89179190909" number while alarm on the first video camera.

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

2. Set device for message delivery and send message to the "89179190909" number while alarm on the first sensor.

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

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

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

TELEMETRY

The **TELEMETRY** object corresponds to the **Telemetry controller** system object.

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

Format of events procedure for the **Telemetry controller** object:

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

Description of events of the **TELEMETRY** object.

Event	**Description**	Comment
LOCKED	Locked	Event is received after the LOCK command (see the following command).
UNLOCKED	Unlocked	Event is received after the UNLOCK command (see the following command)

Operator format to describe actions with the short message service is:

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

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 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 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 selected preset.	tel_prior<>	Priority (1 - low, 2 – medium, 3 – high).
	preset<>	Preset.
"D2OFF" – disable additional dynamic settings for Panasonic ptz video cameras designed to increase quality of analog video signal.	tel_prior<>	Priority (1 - low, 2 – medium, 3 – high).

"D2ON" – enable additional dynamic settings for Panasonic ptz video cameras designed to increase quality of analog video signal.	tel_prior<>	Priority (1 - low, 2 – medium, 3 – high).
"DOWN" – rotate video camera objective 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 position specified on preset.	tel_prior<>	Priority (1 - low, 2 – medium, 3 – high).
	preset<>	Preset.
"HOME" – rotate video camera to 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 objective left.	tel_prior<>	Priority (1 - low, 2 – medium, 3 – high).
"LEFT_DOWN" – rotate video camera objective left and down.	tel_prior<>	Priority (1 - low, 2 – medium, 3 – high).
"LEFT_UP" – rotate video camera objective left and up.	tel_prior<>	Priority (1 - low, 2 – medium, 3 – high).
"PATROL_LEARN" – start procedure of patrol programming performed by record of 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 objective right.	tel_prior<>	Priority (1 - low, 2 – medium, 3 – high).
"RIGHT_DOWN" – rotate video camera objective right and down.	tel_prior<>	Priority (1 - low, 2 – medium, 3 – high).
"RIGHT_UP" – rotate video camera objective right and up.	tel_prior<>	Priority (1 - low, 2 – medium, 3 – high).
"SET_PRESET" – record current position of video camera to the selected preset.	tel_prior<>	Priority (1 - low, 2 – medium, 3 – high).
	preset<>	Preset.
"STOP" – stop video camera rotation.	tel_prior<>	Priority (1 - low, 2 – medium, 3 – high).

"UP" – rotate video camera objective 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 hexadecimal format.	buffer<>	Command in hexadecimal format.
	parent_id<>	ID of **Telemetry controller** parent object. Required parameter.
	tel_prior<>	Priority (1 - low, 2 – medium, 3 – high). Value of parameter is to be more than 0.
LOCK - lock. Switch over telemetry to the LOCKED state for specified time.	tel_prior<>	Priority (1 - low, 2 – medium, 3 – high). Value of parameter is to be more than 0. It is forbidden to perform control commands with lower priority than specified during the blocking time.
	duration<>	Locking duration. Locking in force until UNLOCK command performing if parameter is not specified.
UNLOCK - unlock. Switch over telemetry to the UNLOCKED state for specified time.	-	-

Properties of the **TELEMETRY** object are shown in the table.

Properties of the TELEMETRY object	**Description of properties**
ID<>	Object ID.
PARENT_ID<>	Parent object ID.

The **TELEMETRY** object can be in the following states:

State of the TELEMETRY object	**Description**
LOCKED - locked	Control of telemetry is locked with some priority. It is forbidden to control telemetry with priority higher than specified while locking (see the table below).
UNLOCKED - unlocked	It is allowed to control telemetry with any priority.

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

1. Set autofocus when video camera is armed.

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

2. Rotate video camera to position specified in the first preset while enabling relay.

```
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 2 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>");
}
```

TELEMETRY_EXT

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 appears.

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

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

Event	Description of event	Parameter	Description of parameter	Range of values
"KEY_PRESSED"	Key is pressed	param0<>	Code of pressed key	See Installing and configuring security system components guide.
		device<>	Device on which key is pressed	0 – Main keyboard *AXIS T8312* , 1 - *AXIS T8313 keyboard*
"KEY_RELEASED"	Key is released	param0<>	Code of released key	0..21 for *AXIS T8312*. For *BOSCH KBD-Digital, BOSCH KBD-Universal* and *Panasonic WV-CU950* s ee Installing and configuring security system components guide.
		device<>	Device on which key is released	0 – Main keyboard *AXIS T8312*, 1 – Rotary switch *AXIS T8313*
"MOVED"	Position is changed	param0<>	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

Operator format to describe actions with the short message service is:

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

List of commands and parameters for the **TELEMETRY_EXT** object is presented in the following table:

Command – command description	Parameters	Description
"DRAW_FIGURE" – draw 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, 1 – fill
	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 first line 1 – display text on second line
	text<>	Entered text of line, maximum 20 symbols
	flickering<>	Line consists of 6 symbols determining parameters of text flashing: d1 d2 d3 d4 d5 d6 d1 determines period of flashing : 0 – flashing disabled 1 - period 0.25 sec, symbol is changed by white space 2 - period 0.5 sec, symbol is changed by white space 3 - period 0.75 sec, symbol is changed by white space. 4 - period 1 sec, symbol is changed by white space. 5 - period 0.25 sec, symbol is changed by white space 6 - period 0.5 sec, symbol is changed by white space 7 - period 0.75 sec, symbol is changed by white space 8 - period 1 sec, symbol is changed by white space d2: 1 – symbols from 1 to 4 are flashing, 0 – these symbols are not flashing. d3: 1 – symbols from 5 to 8 are flashing, 0 – these symbols are not flashing. d4: 1 – symbols from 9 to 12 are flashing, 0 – these symbols are not flashing. d5: 1 – symbols from 13 to 16 are flashing, 0 – these symbols are not flashing. d6: 1 – symbols from 17 to 20 are flashing, 0 – these symbols 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-CU950panel*	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 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

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

1. 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");
    }
}
```

MACRO

The **MACRO** object corresponds to the **Macros** system object.

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

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

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

Event	Description
"RUN"	Action is performed

Operator format to describe actions with the macros is:

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

List of commands and parameters for the **MACRO** object is presented in the following table:

Command – command description	Parameters	Description

"RUN" – perform action	-	-
"SETUP" – set parameters for 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.

Examples of using events and reaction of the MACRO object:

1. Set the current position of video camera to preset while performing the macro 1.

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

2. Perform macro 2 in case of camera is armed.

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

TIME_ZONE

The **TIME_ZONE** object corresponds to the **Time zones** system object.

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

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

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

Event	Description
"ACTIVATE"	Start.
"DEACTIVATE"	End.

Operator format to describe actions with the time zones is:

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

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.

Check function of the **TIME_ZONE** object state:

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

The **TIME_ZONE** object can be in the following states:

State of the TIME_ZONE object	Description
"ACTIVATE"	Active.
"INACTIVE"	Inactive.

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

1. Display video image from the camera №1 on the monitor while activation of the first time zone.

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

SSS_WATCHDOG

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 appears.

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

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

Event	Description
"RESTART_EXCEEDED"	Number of module restart is exceeded.

"RESTART_PROCESS"	Module restart.

Operator format to describe actions with the system restart service is:

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

List of commands and parameters for the **SSS_WATCHDOG** object is presented in the following table:

Command – command description	**Parameters**	**Description**
"SETUP" – set up 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 ITV 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.

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

1. Activate the third camera on the monitor №5 while module restarting.

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

SLAVE

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 appears.

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

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

Events	Description	Comment
CONNECTED	Connecting.	Event is generated when some Client is connected to the Server.
DISCONNECTED	Disconnecting.	Event is generated when some 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 object tree is more than specified in the key.
KEY_UPDATED	Key updated.	
PROTOCOL_RCVD	Protocol received.	
REBUILD_IN_START	Start of archive reindexing.	
REBUILD_IN_STOP	End of archive reindexing.	
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 (while 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 mounted.	
DISC_UNMOUNT	Disc unmounted.	
ARCHIVE_DEPTH	Archive depth.	Event is generated in the midnight and contains information about archive depth on all disks in hours (e. g. depth<> parameter). To raise event manually use GET_DEPTH reaction. Archive depth in format Days:Hours is specified in the **Ad ditional information** field of the Event viewer while event displaying. 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 *Intelle ct*, 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.

Operator format to describe actions with the computer object is:

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

List of commands and parameters for the **SLAVE** object is presented in the following table:

Command – command description	**Parameters**	**Description**
"**SETUP**" **–** set up parameters for 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 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\ITV\INTELLECT\ (HK	-	-

EY_LOCAL_MACHINE \Software\Wow6432Node\ITV\INTELLECT 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<>	Disc 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.

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

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

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

2. Get the archive depth by Camera 1 using the 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 parameter of the **param0<>** event.

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. The event procedure format for the **Display** object looks like this:

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

Description of events of the **DISPLAY** object:

Event	Description
ACTIVATE	Display is activated
DEACTIVATE	Display is deactivated

Operator format to describe actions with the screen is:

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

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 computer on which display is to be shown.
DEACTIVATE – hide display.	macro_slave_id<>	Name of computer on which display is to be hidden**.**

> **Note.**
>
> If «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

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

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

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

GATE

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 appears.

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

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

Events	Description	Comment
GATE_LOW_FPS	Input speed on the gate is reduced	
ACTIVE	Gate is active	Event is generated when list of worked cameras corresponds to the list of Videogate configuration.
INACTIVE	Gate is inactive	Event is generated when there are no requests for video flows through the Videogate.
ACTIVE_PART	Partial work of gate	Event is generated if number of worked cameras is less than in the gate list.

Example. Send corresponding messages to all system cores while reducing the input speed.

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

CAM_VMDA_DETECTOR

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 appears.

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

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

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 (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" type object. Allows determining quantity of people in the group. For other object types it is equal to -1.
"ALARM_END"	End of alarm.		

Operator format to describe actions with the VMDA detection is:

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

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.	-	-

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

Arm the VMDA Detection 2 while performing macro 1:

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

ARCH

The **ARCH** object corresponds to the **Archive** system object.

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

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

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

Events	Description	Comment
ACTIVE	Archive is active	Event is generated when list of cameras video from which is archived corresponds to the list of Archive configuration
INACTIVE	Archive is inactive	Event is generated when archiving through the Archive is not performed.
ACTIVE_PART	Partial work of archive	Event is generated when archiving is enabled not for all cameras specified in the list of Archive.

Example. Send corresponding message to all cores of system if archiving through the Archive 1 is not performed.

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

CORE

The CORE object is the global static object realized methods used to control state and to manage system objects of the Intellect software package. The extended possibilities for working with the CORE object are given when using scripts on JScript programming language – see the Programming Guide (JScript).

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

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

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

Event	Description
DO_REACT	Event triggers reaction of some object in the system. Description of action which is to be performed is forwarded in the action parameter of this event. Examples of values of action parameter: SET_MARKRECT – sends when face recognizing on the video image; DEL_MARKRECT – sends when face disappearing from the video image. Can be other parameters of the event which can be monitored using the Debug window (see Programming Guide (JScript), section The Debug window). For example, if value of action parameter is SET_MARKRECT than number of camera on video image of which face is recognized is forwarding in the param5_val parameter. It shows name of parameter forwarded in the param5_name parameter. For the DEL_MARKRECT value the camera number is forwarding in the param0_val parameter.
SLAVE_CHANGED	Event is generated when Failover becomes active. It consists of the following parameters: old_slave_id – ID of the **Computer** object where cameras are moved from. new_slave_id – ID of the **Computer** object where cameras are moved to. CAM<n1, n2, ... > – where n1, n2, etc. are IDs of cameras moved to another parent **Computer** object. For example, CAM<4,6,7> – move cameras with IDs 4, 6, 7.

Example.

When a face appears in the frame show the video from corresponding camera on the Monitor 2. When the face disappears – hide video from corresponding camera on the 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+">");
        ]
    }
}
```

JOYSTICK

The **JOYSTICK** object does not correspond to any system object.

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

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

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

Description of events of the **JOYSTICK** object.

Events	Parameter	Parameter description
"KEY_PRESSED" – Key pressed	button<>	The code of the key pressed

TITLEVIEWER

The **TITLEVIEWER** object corresponds to the **Search by titles** system object.

The events given in the table come from the **TITLEVIEWER** object. When the corresponding event happens, the procedures are run. The event procedure format for the object **Search by titles** system object:

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

The events coming from the **TITLEVIEWER** object are given in the table.

Event	Event description	Parameters	Parameters description	Comment
GO_VIDEO	Video request	<cam>	The ID of the camera where the titles were found	The event is generated when left double-clicking the search result line.
		<date>	Date	
		<time>	Time	

Example.

When double-clicking the search result line in the **Search by titles** system object window, one can 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>");
}
```

MAP

The **MAP** object corresponds to the **Map** system object.

Operator format to describe actions with the map:

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

The list of commands and parameters for the **MAP object is given 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 ID. Can be left empty. If the object type is not set, then objects of all types are hidden/shown.

	objid<>	Object ID. Can be left empty. If the ID is not set, then all objects of 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 in pixels along the X axis.
	y<>	New coordinate of the top left corner of the object icon on the map layer in pixels along the Y axis.
	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 its name.
INSCRIBE – inscribe to window	-	-
SHOW_MINIMAP – show a minimap	x<>	The coordinate of the upper-left corner of the minimap along the X axis in pixels.
	y<>	The coordinate of the upper-left corner of the minimap along the Y axis in pixels.
	w<>	Width of minimap in pixels.
	h<>	Height of minimap in pixels.
	monitor<>	Monitor ID.
	__slave_id<>	Net name.

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>");
}
```

FAILOVER

The **FAILOVER** object corresponds to the **Failover system object**.

Events from the **FAILOVER** object are given in the table. Procedures are run when a corresponding event happens. The event procedure format for the **Failover** object:

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

Description of events from the **FAILOVER object**:

Event	Description
START	Objects are moved to a backup Server.
STOP	Objects are back to the main Server.

Example uses can be found in the Examples of scripts in JScript language section in Programming Guide (JScript).

OPERATORPROTOCOL

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 happens. The format of the event procedure for the **Operator protocol** object:

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

Event	Event description
ACTIVATE_LEFT	Operator left-clicked the event cell in the Operator protocol
ACTIVATE_RIGHT	Operator right-clicked the event cell in the Operator protocol

The operator format for description of actions with the **Operator protocol** object:

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

The list of commands and parameters for the **OPERATORPROTOCOL** object is given in the table.

Command – its description	Parameters	Parameter description
DEL_ALARM	objtype<>	Object type (e.g., CAM, GRELE, etc.)
	objid<>	Object ID
	options<>	Possible values: first – delete first alarm last – delete last alarm all or empty – delete all alarms
HIDE_BUTTON – hide buttons used to assign status to event.	button<>	Names of buttons separated by comma: alarm – Alarm situation suspicious – Suspicious situation false – False triggering Example of setting parameter: `button<alarm,suspicious,false>`
	hide<>	1 – hide buttons listed in the button parameter 0 – show buttons listed in the button parameter
	objtype<>	Object type
	objaction<>	Event type
	objid<>	Object ID

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

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

Example 2. Hide the Alarm, Suspicious and False buttons for the Disarm event from Camera 12 in the Operator protocol 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>");
}
```

PERSON

The **PERSON** object corresponds to the **User** system object.
The events presented in the table come from the **PERSON** object. Procedures are run when the corresponding event occurs. The event procedure format for the **Camera** object looks like this:

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

Description of events of the **PERSON object**:

Event	Description
"REGISTERED"	User enters the system
"UNREGISTERED"	User exits the system

Programming Guide. Conclusion

More detailed information on the Intellect software package is presented in the documents titled:

1. Administrator's Guide;
2. Operator's Guide;
3. Installing and configuring security system components guide;
4. Programming Guide (JScript).

If you have faced difficulties and problems while operating the given software product, you are welcome to contact us. However, before addressing 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 occurrence?
3. Which conditions gave rise to the problem?

Remember, that 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 proposals and suggestions on how to improve our software and documentation.

You are welcome to send your comments and suggestions regarding this Guide to the AxxonSoft Training and documentation development division (documentation@itv.ru).

Appendix 1. Priorities of start and stop recording commands

Start and stop recording commands can have different priorities in *Intellect*. The priority of start/stop recording commands is set by the priority<> parameter of the REC and REC_STOP reactions. 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 the detection tool triggering, the priority is not set. The table shows the behavior of *Intellect* when various ways to start/stop recording are in use.

Start/stop recording 1	**Start/stop recording 2**	**Behavior**
Start recording at CAM|1|REC| reaction, stop recording at CAM|1|REC_STOP| reaction	Initiated by the operator using the camera context menu (start/stop recording)	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)	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)	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)	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)	Start/stop recording is initiated by the detection tool (for example, 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, 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, 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, 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 param_id and param_value values for SET_IPINT_PARAM reaction

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

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

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

Note.

In most cases, the name of driver is the same as the name of the vendor of IP device. Contact AxxonSoft support team in order to obtain more specific information about the driver name.

3. In the file find the name of required model, for instance 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. One more occurrence of the **id** value of this parameter is to be found 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 SET_IPINT_PARAM reaction:

1. For **Camera** object:
   DoReact("CAM", "1","SET_IPINT_PARAM","param_id<daynight>,param_value<on>");
2. For **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 SET_IPINT_PARAM reaction, the multistream mode is to be active - see Configuration of multistream video section of Administrator's Guide. 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 Intellect software (go to Documentation Drivers Pack page).

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 **<videoStreamingRef>** tags. There should be more than one stream.