



Guide for creating scripts (programming)

Last update 17/11/2023

Table of contents

1	Guide for creating scripts. Introduction.....	9
2	The Program object. Programming using the embedded language of Intellect	10
3	The Script object. Programming using the JScript language	11
4	Description of events and reactions of system objects.....	12
5	Description of the object model in Intellect	13
6	Programming Guide. Conclusion	15
7	Guide for creating scripts. Introduction.....	16
7.1	Methods of setting logical interactions between objects in Intellect	16
8	The Program object. Programming using the embedded language of Intellect	17
8.1	Programming tools in Intellect.....	17
8.1.1	The Program system object.....	17
8.1.2	Debug window	18
8.1.3	Syntax analyser	18
8.1.4	Recommended procedure of writing programs	19
8.1.4.1	Setting a general task	20
8.1.4.2	Outlining the task into subtasks.....	20
8.1.4.3	Writing subtasks and debugging them	20
8.1.4.4	Finding and fixing bugs.....	20
8.2	Description of syntax	21
8.2.1	Description of variables	21
8.2.2	Description of procedures	21
8.2.2.1	Standard procedures	21
8.2.2.2	Creating custom procedures	23
8.2.3	Description of operators.....	23
8.2.4	Operators and expressions.....	26
8.2.5	Description of functions	28
8.2.6	Examples of scripts in the embedded language.....	38
8.2.6.1	Example 1.	39
8.2.6.2	Example 2.	39
8.2.6.3	Example 3.	39
8.2.6.4	Example 4.	39
8.2.6.5	Example 5.	40

8.2.6.6	Example 6	40
8.2.6.7	Example 7	41
8.2.6.8	Example 8	42
8.2.6.9	Example 9	43
8.2.6.10	Example 10	44
8.2.6.11	Example 11	45
8.2.6.12	Example 12	45
8.3	Appendix 1. Priorities of start and stop recording commands	46
8.4	Appendix 2. Defining param_id and param_value values for SET_IPINT_PARAM reaction	47
9	The Script object. Programming using the JScript language	50
9.1	Programming in JScript	50
9.2	Appendix 1. Description of the Editor-Debugger utility	50
9.3	Appendix 2. Creating virtual objects with ability to set events, reactions and states	50
9.4	Programming Guide (JScript). Conclusion	51
9.5	Programming in JScript	51
9.5.1	JScript functionality in Intellect	51
9.5.2	Programming tools	51
9.5.2.1	The Script object	51
9.5.2.2	The Editor-Debugger utility	53
9.5.2.3	The Debug window	55
9.5.2.3.1	Enabling the Debug window	55
9.5.2.3.2	Working with Debug window	56
9.5.2.3.2.1	Copying information on event or reaction to the clipboard	57
9.5.2.3.2.2	Highlighting messages	58
9.5.2.3.2.3	Event and reaction filter	59
9.5.2.3.2.4	Searching for events and reactions	60
9.5.2.3.2.5	Clearing the Debug window	61
9.5.2.4	Getting the list of object names, reactions and events in INTELLECT™	61
9.5.3	Operations with scripts	64
9.5.3.1	Creating a script	64
9.5.3.1.1	Creating the Script object	64
9.5.3.1.2	Creating and editing a script	65
9.5.3.1.2.1	Features when working with a script	67
9.5.3.1.3	Debugging a script	67
9.5.3.2	Saving a script	67

9.5.3.3	Deleting a script	67
9.5.3.4	Search text in script	68
9.5.3.5	Replace text in script.....	68
9.5.4	Creating your first script	69
9.5.5	Script debugging.....	74
9.5.5.1	Script debugging features	74
9.5.5.2	Creating and using test events	74
9.5.5.2.1	Creating test events	74
9.5.5.2.2	Running the script with a test event	76
9.5.5.3	Using debugger windows of the Editor-Debugger utility.....	76
9.5.5.3.1	Displaying messages about starting, verifying, changing and executing scripts in the debugger windows	76
9.5.5.3.2	Viewing the script messages.....	78
9.5.5.4	Using third-party debugger programs	79
9.5.6	Examples of scripts in JScript language	80
9.5.6.1	Example 1. Visualisation of operating the Queue length detection in the Video Surveillance monitor.....	81
9.5.6.2	Example 2. Visualisation of operating the People Counter detection in the Video Surveillance monitor	82
9.5.6.3	Example 3. Setting the text for displaying on the map	83
9.5.6.4	Example 4. Displaying camera on the monitor by clicking the button on the remote control.....	83
9.5.6.5	Example 5. Sending command to camera using the camera HTTPAPI.....	84
9.5.6.6	Example 6. Script for framing abandoned objects in live video	85
9.5.6.7	Example 7. Using START and STOP events for the Failover Service	86
9.5.6.8	Example 8. Using the embedded visitors counter on Bosch FLEXIDOME IP dynamic 7000 VR IP camera	86
9.5.6.9	Example 9. Sending e-mail with HTML markup.....	86
9.5.6.10	Example 10. Creating test users	87
9.5.6.11	Example 11. Changing the status of the event in the Incident manager.....	88
9.6	Appendix 1. Description of the Editor-Debugger utility	89
9.6.1	The purpose of the Editor-Debugger utility.....	89
9.6.2	The interface of the Editor-Debugger utility.....	89
9.6.2.1	The Editor-Debugger window	89
9.6.2.2	The Script Debug/Edit tab	89
9.6.2.2.1	Description of the Script Debug/Edit tab.....	89
9.6.2.2.2	The Script object panel in the Script Debug/Edit tab.....	90
9.6.2.3	The Script Messages tab	91
9.6.2.3.1	Description of the Script Messages tab.....	91
9.6.2.3.2	The Script object panel in the Script Messages tab	92

9.6.2.4	Main menu	93
9.6.2.4.1	Description of the main menu	93
9.6.2.4.2	The elements in the File menu	94
9.6.2.4.3	The elements in the View menu	94
9.6.2.4.4	The elements of the Debug and edit menu	95
9.6.2.4.5	The Message list menu elements	95
9.6.2.5	The Filter dialog window	95
9.6.2.6	The Color dialog window	96
9.6.2.7	The toolbar of the Editor-Debugger utility	97
9.7	Appendix 2. Creating virtual objects with ability to set events, reactions and states	99
9.7.1	Purpose of virtual objects and their implementation in Intellect	99
9.7.2	How to create a virtual object	99
9.7.2.1	DBI file preparation	99
9.7.2.2	DDI file preparation	100
9.7.2.3	XML file preparation	103
9.7.2.4	Creating and using a virtual object in Intellect	104
9.7.2.4.1	Displaying on the map	105
9.7.2.4.2	Using in macros	106
9.7.2.4.3	Sample program in JScript to change the state of a virtual object	106
9.8	Programming Guide (JScript). Conclusion	107
10	Description of events and reactions of system objects.....	108
10.1	GRABBER	108
10.2	CAM	112
10.3	MONITOR	121
10.4	PLAYER.....	130
10.5	OLXA_LINE.....	131
10.6	DIALOG.....	133
10.7	MMS.....	134
10.8	MAIL_MESSAGE	135
10.9	VMS	136
10.10	GRELE.....	137
10.11	GRAY.....	139
10.12	VNS.....	141
10.13	SMS	143

10.14	TELEMETRY.....	144
10.15	TELEMETRY_EXT	148
10.16	MACRO	151
10.17	TIME_ZONE.....	152
10.18	SSS_WATCHDOG	153
10.19	SLAVE	154
10.20	DISPLAY.....	158
10.21	GATE.....	159
10.22	CAM_VMDA_DETECTOR	160
10.23	ARCH	161
10.24	CORE	162
10.25	IPJOYSTICK.....	163
10.26	TITLEVIEWER	163
10.27	MAP	163
10.28	FAILOVER.....	166
10.29	OPERATORPROTOCOL.....	167
10.30	PERSON	168
10.31	EVENT_VIEWER.....	169
10.32	CAM_TITLE.....	169
10.33	CAM_FACECAPTURE.....	170
10.34	IPSTORAGE	171
10.35	TELEGRAM	171
10.36	BACNET	173
10.36.1	Examples	174
10.37	CAM_IP_DETECTOR.....	176
10.38	SIP_TERMINAL.....	177
10.39	INC_MANAGER.....	178
10.40	INC_SERVER.....	179
11	Description of the object model in Intellect	181
11.1	The Core object and its built-in methods	181
11.1.1	The Core object	181
11.1.2	The SetObjectParam method.....	181
11.1.3	The SetObjectState method	182

11.1.4	The DebugLogString method	182
11.1.5	The Base64Decode method.....	183
11.1.6	The Sleep method	183
11.1.7	The Itv_var method.....	184
11.1.8	The Int_var method	184
11.1.9	The GetObjectParentType method	185
11.1.10	The GetIPAddress method.....	186
11.1.11	The GetObjectName method	186
11.1.12	The GetObjectState method.....	187
11.1.13	The GetObjectParam method	188
11.1.14	The GetObjectParentId method.....	188
11.1.15	The DoReactStr method	189
11.1.16	The DoReact method	190
11.1.17	The DoReactSetupCore method	191
11.1.18	The DoReactSetup method	192
11.1.19	The DoReactGlobal method	192
11.1.20	The NotifyEventStr method.....	193
11.1.21	The NotifyEvent method.....	194
11.1.22	The NotifyEventGlobal method.....	195
11.1.23	The CreateMsg method.....	195
11.1.24	The Lock and Unlock methods.....	196
11.1.25	The IsAvailableObject method	197
11.1.26	The GetUserId method.....	198
11.1.27	The GetEventDescription method.....	199
11.1.28	The GetObjectIdByParam method.....	199
11.1.29	The SaveToFile method	200
11.1.30	The GetLinkedObjects method	200
11.1.31	The WriteIni method	201
11.1.32	The ReadIni method.....	201
11.1.33	The AddIni method	201
11.1.34	The SetTimer method	202
11.1.35	The KillTimer method	203
11.1.36	The GetObjectChildIds method.....	203
11.1.37	The Base64EncodeFile method.....	203
11.1.38	The Base64EncodeW method.....	204

11.1.39	The run_cmd and run_cmd_timeout methods.....	204
11.1.40	The WriteIniAny method	205
11.1.41	The ReadIniAny method	205
11.1.42	The AddIniAny method	206
11.2	The MsgObject and Event objects and their built-in methods and properties.....	206
11.2.1	The MsgObject and Event objects	206
11.2.2	The GetSourceType method	207
11.2.3	The GetSourceId method	208
11.2.4	The GetAction method.....	208
11.2.5	The GetParam method	208
11.2.6	The SetParam method.....	209
11.2.7	The MsgToString method	209
11.2.8	The StringToMsg method	209
11.2.9	The StringToParams method	210
11.2.10	The Clone method.....	211
11.2.11	The GetObjectIds method	211
11.2.12	The GetObjectParams method.....	212
11.2.13	The SourceType property	212
11.2.14	The SourceId property	213
11.2.15	The Action property	213
12	Programming Guide. Conclusion	214

1 Guide for creating scripts. Introduction

2 The Program object. Programming using the embedded language of Intellect

- Programming tools in Intellect
 - The Program system object
 - Debug window
 - Syntax analyser
 - Recommended procedure of writing programs
- Description of syntax
 - Description of variables
 - Description of procedures
 - Standard procedures
 - Creating custom procedures
 - Description of operators
 - Operators and expressions
 - Description of functions
 - Examples of scripts in the embedded language
- Appendix 1. Priorities of start and stop recording commands
- Appendix 2. Defining param_id and param_value values for SET_IPINT_PARAM reaction

3 The Script object. Programming using the JScript language

- Programming in JScript
 - JScript functionality in Intellect
 - Programming tools
 - The Script object
 - The Editor-Debugger utility
 - The Debug window
 - Getting the list of object names, reactions and events in INTELLECT™
 - Operations with scripts
 - Creating a script
 - Saving a script
 - Deleting a script
 - Search text in script
 - Replace text in script
 - Creating your first script
 - Script debugging
 - Script debugging features
 - Creating and using test events
 - Using debugger windows of the Editor-Debugger utility
 - Using third-party debugger programs
 - Examples of scripts in JScript language
- Appendix 1. Description of the Editor-Debugger utility
 - The purpose of the Editor-Debugger utility
 - The interface of the Editor-Debugger utility
 - The Editor-Debugger window
 - The Script Debug/Edit tab
 - The Script Messages tab
 - Main menu
 - The Filter dialog window
 - The Color dialog window
 - The toolbar of the Editor-Debugger utility
- Appendix 2. Creating virtual objects with ability to set events, reactions and states
 - Purpose of virtual objects and their implementation in Intellect
 - How to create a virtual object
 - DBI file preparation
 - DDI file preparation
 - XML file preparation
 - Creating and using a virtual object in Intellect
- Programming Guide (JScript). Conclusion

4 Description of events and reactions of system objects

- GRABBER
- CAM
- MONITOR
- PLAYER
- OLXA_LINE
- DIALOG
- MMS
- MAIL_MESSAGE
- VMS
- GRELE
- GRAY
- VNS
- SMS
- TELEMETRY
- TELEMETRY_EXT
- MACRO
- TIME_ZONE
- SSS_WATCHDOG
- SLAVE
- DISPLAY
- GATE
- CAM_VMDA_DETECTOR
- ARCH
- CORE
- IPJOYSTICK
- TITLEVIEWER
- MAP
- FAILOVER
- OPERATORPROTOCOL
- PERSON
- EVENT_VIEWER
- CAM_TITLE
- CAM_FACECAPTURE
- IPSTORAGE
- TELEGRAM
- BACNET
- CAM_IP_DETECTOR
- SIP_TERMINAL
- INC_MANAGER
- INC_SERVER

5 Description of the object model in Intellect

- The Core object and its built-in methods
 - The Core object
 - The SetObjectParam method
 - The SetObjectState method
 - The DebugLogString method
 - The Base64Decode method
 - The Sleep method
 - The Itv_var method
 - The Int_var method
 - The GetObjectParentType method
 - The GetIPAddress method
 - The GetObjectName method
 - The GetObjectState method
 - The GetObjectParam method
 - The GetObjectParentId method
 - The DoReactStr method
 - The DoReact method
 - The DoReactSetupCore method
 - The DoReactSetup method
 - The DoReactGlobal method
 - The NotifyEventStr method
 - The NotifyEvent method
 - The NotifyEventGlobal method
 - The CreateMsg method
 - The Lock and Unlock methods
 - The IsAvailableObject method
 - The GetUserId method
 - The GetEventDescription method
 - The GetObjectIdByParam method
 - The SaveToFile method
 - The GetLinkedObjects method
 - The Writeln method
 - The ReadIni method
 - The AddIni method
 - The SetTimer method
 - The KillTimer method
 - The GetObjectChildIds method
 - The Base64EncodeFile method
 - The Base64EncodeW method
 - The run_cmd and run_cmd_timeout methods
 - The WritelnAny method
 - The ReadIniAny method
 - The AddIniAny method
- The MsgObject and Event objects and their built-in methods and properties
 - The MsgObject and Event objects
 - The GetSourceType method
 - The GetSourceId method
 - The GetAction method
 - The GetParam method
 - The SetParam method
 - The MsgToString method
 - The StringToMsg method
 - The StringToParams method
 - The Clone method
 - The GetObjectIds method
 - The GetObjectParams method

- The SourceType property
- The SourceId property
- The Action property

6 Programming Guide. Conclusion

7 Guide for creating scripts. Introduction

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

There are two options for writing scripts in *Intellect*:

1. **In the embedded programming language.** You can use the **Program** system object of the **Programming** tab—see [The Program object. Programming using the embedded language of Intellect.](#)
2. **In the JScript language.** You can use the **Script** system object of the **Programming** tab—see [The Script object. Programming using the JScript language.](#)

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

This guide had the following information:

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

The scripts use:

- events, reactions and commands of *Intellect* objects. The main ones are described in [Description of events and reactions of system objects.](#)
- the Core, MsgObject, Event objects and their embedded methods described in [Description of the object model in Intellect.](#)

7.1 Methods of setting logical interactions between objects in Intellect

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

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

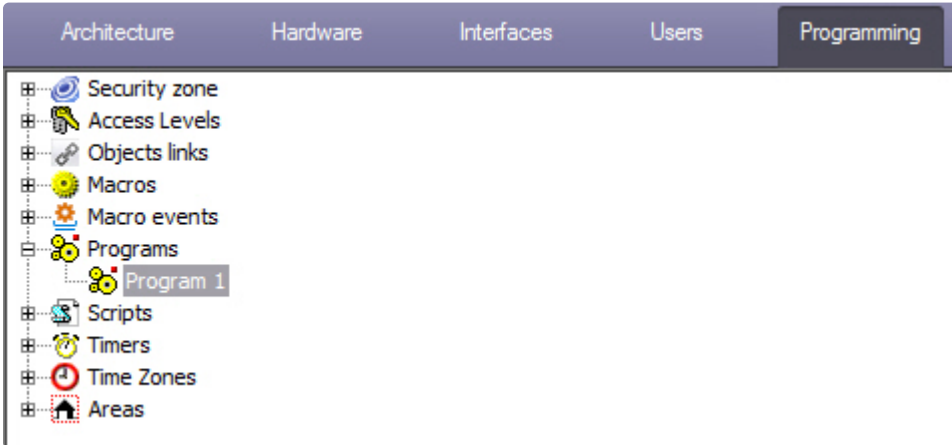
8 The Program object. Programming using the embedded language of Intellect

8.1 Programming tools in Intellect

8.1.1 The Program system object

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

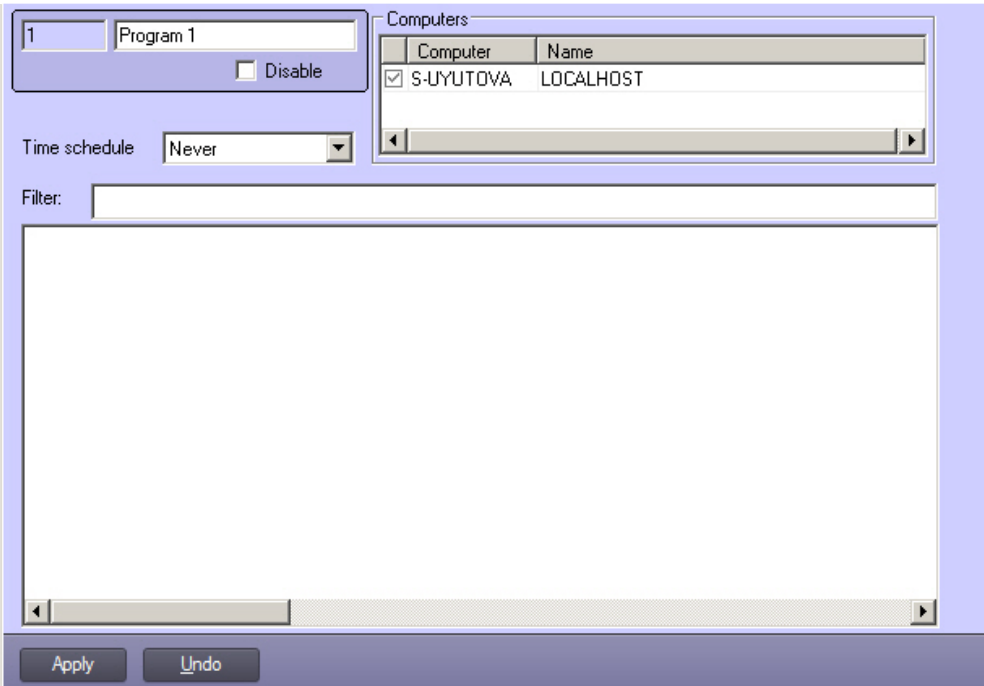
The **Program** system object is created on the basis of the **Programs** object on the **Programming** tab of the **System settings** dialog window.



Attention!

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

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



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

Note

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

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

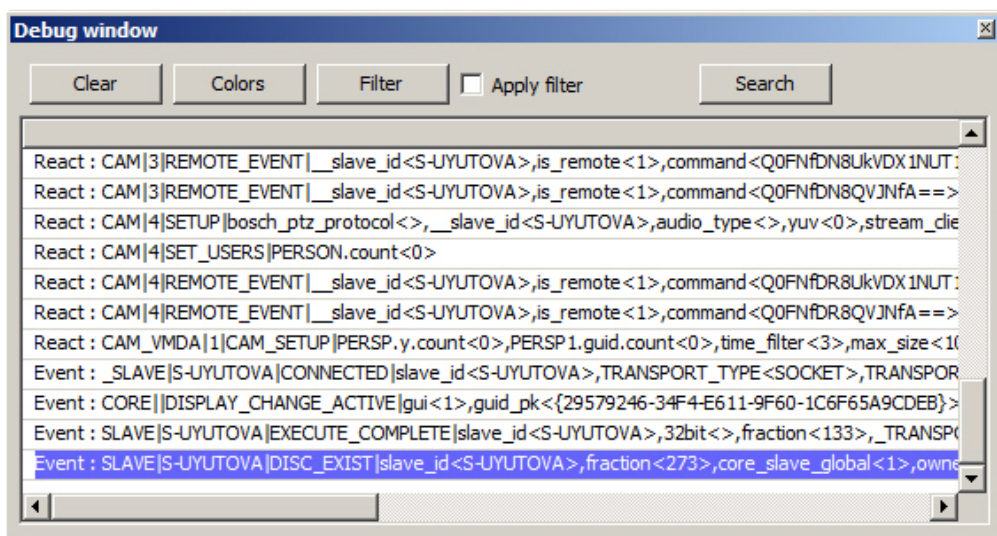
There is the word processor on the settings panel of the **Program** system object. It is used for writing and editing the program code.

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

8.1.2 Debug window

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

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



By default, the **Debug window** isn't available. You can enable the **Debug window** by using the *tweak*.exe utility (see [The Debug window](#) section of [The Script object. Programming using the JScript language](#)).

8.1.3 Syntax analyser

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

```

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

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

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

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

```

To change the font size, use the key combinations:

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

```

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

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

```

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

```

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

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

```

8.1.4 Recommended procedure of writing programs

On the page:

- [Setting a general task](#)
- [Outlining the task into subtasks](#)

- Finding and fixing bugs

1. Setting a general task.
2. Outlining the task into subtasks.
3. Writing subtasks and debugging them.
4. Finding and fixing bugs.

8.1.4.1 Setting a general task

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

8.1.4.2 Outlining the task into subtasks

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

8.1.4.3 Writing subtasks and debugging them

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

8.1.4.4 Finding and fixing bugs

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



Note

If the syntax is correct, but the program still doesn't work or works with errors, it is recommended to rewrite the program as a script in JScript (see [The Script object. Programming using the JScript language](#)).

8.2 Description of syntax

Script consists of the set of procedures.

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

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

8.2.1 Description of variables

All variables in the system are string variables.

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

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

8.2.2 Description of procedures

8.2.2.1 Standard procedures

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

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

Example of use:

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

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

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

Examples of use:

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

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

```

    //the code will be executed every day at 11:11:30
}

```

```

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

```

```

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

```

```

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

```

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

Examples use:

```

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

```

```

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

```

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

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

Examples of use:

```

OnEvent("GRAY","1","ON")      // will be executed when closing sensor 1
{
    i=1;
    i=i+1;    //as variables are string, then the sum will be 11
    j=1;
    j=str(j+1); // str is a number-to-string conversion function. Inside the str
}

```

```

        //function all string variables (if any) are converted into 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
    }
}

```

8.2.2.2 Creating custom procedures

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

```

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

```

Attention!

The names of parameters must consist of one uppercase character.

Examples of use:

```

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

OnEvent("MACRO",N,"RUN")
{
    a1=N;
    a2="Macro";
    ProcedureName(a2,a1);
}

```

8.2.3 Description of operators

The list of operators used to describe actions:

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

```

OnEvent("GRAY","1","ON")
{

```

```

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

```

2. DoCommand(command line)—run the command line

Examples use:

```

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

```

3. Wait(number of seconds)—wait for N seconds;

Sleep(number of milliseconds)—wait for N milliseconds.

Await operators must be in a single thread. Single thread must be inside square brackets.

Example. When Sensor 1 is closed, Relay 1 is closed for five seconds.

```

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

```

4. Function to check the object state:

CheckState(object type, number, state)—the result is 1, if the state of an object is factually accurate, otherwise 0.

Expressions can be used as parameters. Constant values are quoted.

Example. Check the state of camera 2 when closing sensor 1 and if the state is “Alarmed“, then close relay 1

```

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

```

5. Conditional operator:

```

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

```

else {} part can be absent.

Example of use:

```

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

```

```

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

```

6. For operator:

```

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

```

Expression1 is executed at the beginning of the loop; loop body is executed if expression2 is true; expression3 is executed after each execution of the loop body.

Example. When sensor1 is closed, relay1 is closed and opened every second and it will happen 10 times.

```

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

```

7. DoReactGlobal(object type, number, state)—function that creates reactions of system objects. Meanwhile, the created reaction is sent to all cores connected over the network.

Example. When running macro 1, camera 1 is armed.

```

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

```

8. NotifyEventGlobal(object type, number, state)—function that creates system events. Meanwhile, the created events are sent to all cores connected over the network.

Example. When running macro 1, create event “Recording” for camera 1. The command is sent to all cores as an event in order to be logged.

```

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

```

Note

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

8.2.4 Operators and expressions

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

Operator	General description, example of use
Comparison operators	
>	Comparison operator—greater See example in Description of operators
<	Comparison operator—less See example in Description of operators
Arithmetic operators	
+	Addition operator. Example of use: <pre>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 }</pre>
-	Subtraction operator. Example of use: <pre>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 }</pre>
*	Multiplication operator. Example of use: <pre>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 }</pre>
/	Division operator. Example of use: <pre>OnEvent ("MACRO","1","RUN")</pre>

Operator	General description, example of use
	<pre data-bbox="550 271 1149 456"> { x=5; y=10; i=x/y; // divide integers 5/10=0.5 e=str(x/y); // divide integers 5/10=0.5 } </pre>
%	<p data-bbox="520 517 962 544">Remainder after integer division. Example of use.</p> <pre data-bbox="550 600 1437 853"> 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 } </pre>
()	<p data-bbox="520 913 941 940">Group of arithmetic operators. Example of use.</p> <pre data-bbox="550 996 941 1122"> OnEvent ("MACRO","1","RUN") { x=100/((5*8)/1.028); } </pre>
Logical operators	
&&	<p data-bbox="520 1256 863 1283">Logical AND operator. Example of use:</p> <pre data-bbox="550 1339 986 1787"> OnEvent ("MACRO","1","RUN") { a=1; b=2; z=3; if((a<b)&&(b<z)) { y=1; //if false, then else } else { x=0; } } </pre>
!	<p data-bbox="520 1848 906 1874">Logical inversion operator. Example of use:</p> <pre data-bbox="550 1930 954 1986"> OnEvent ("CAM",N,"MD_START") { </pre>

Operator	General description, example of use
	<pre> if(!(strequal(N,"1",)) { DoReact("GRELE","1","ON) } else { DoReact("GRELE","2","ON) } } </pre>

8.2.5 Description of functions

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

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

Functions (The number of executable parameters is specified in square brackets)	General description, example of use
	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: $y=\text{acos}(x)$; where y —function value (in radians), x —argument Example: $y=\text{acos}(0.55)$ 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: $y=\text{atan}(x)$; where y —function value (in radians), x —argument Example: $y=\text{atan}(1.2)$ Event received: 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: $y=\text{sinh}(x)$; where y —function value, x —argument of function Example: $y=\text{sinh}(0.8)$ 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: $y=\text{cosh}(x)$; where y —function value, x —argument of function Example: $y=\text{cosh}(0.35)$ Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<0.336376>,name<y>,time<17:25:25>,date<21-09-04>
tanh[1]	Trigonometric function for an angle calculation. Format: $y=\text{tanh}(x)$; where y —function value, x —argument of function Example: $y=\text{tanh}(0.35)$ Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<1.419068>,name<y>,time<17:25:25>,date<21-09-04>
exp[1]	Returns the value of the e^x function, where x —specified numeric expression. Format: $y=\text{exp}(x)$; where y —function value, x —argument Example: $y=\text{exp}(1.65)$ Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<5.20698>,name<y>,time<17:39:22>,date<21-09-04>

Functions (The number of executable parameters is specified in square brackets)	General description, example of use
log[1]	<p>Returns the natural logarithm (base-e) of the specified numeric expression.</p> <p>Format: $y=\log(x)$; where y—function value, x—argument</p> <p>Example: $y=\log(0.65)$</p> <p>Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<-0.430783>,name<y>,time<17:43:22>,date<21-09-04></p>
log10[1]	<p>Returns the common logarithm (base-10) of the specified numeric expression.</p> <p>Format: $y=\log_{10}(x)$; where y—function value, x—argument</p> <p>Example: $y=\log_{10}(0.05)$</p> <p>Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<-1.30103>,name<y>,time<17:46:28>,date<21-09-04></p>
sqrt[1]	<p>Returns the square root of the specified numeric expression.</p> <p>Format: $y=\sqrt{x}$; where y—function value, x—argument</p> <p>Example: $y=\sqrt{9}$</p> <p>Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<3>,name<y>,time<17:25:25>,date<21-09-04></p>
abs[1]	<p>The abs function returns the absolute value of the argument.</p> <p>Format: $y=\text{abs}(x)$; where y—function value, x—argument</p> <p>Example: $y=\text{abs}(-1)$</p> <p>Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<1>,name<y>,time<13:39:37>,date<22-09-04></p>
deg[1]	<p>Trigonometric function for an angle calculation. Returns the grade measure.</p> <p>Format: $y=\text{deg}(x)$; where y—function value in grades, x—argument value in radians</p> <p>Example: $y=\text{deg}(3.14)$</p> <p>Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<179.908748>,name<y>,time<13:13:51>,date<22-09-04></p>
rad[1]	<p>Trigonometric function for an angle calculation.</p> <p>Format: $y=\text{rad}(x)$; where y—function value in radians, x—argument value in grades</p> <p>Example: $y=\text{rad}(180)$</p> <p>Event received: Event : CORE VAR_CHANGED value<3.141593>,name<y>,time<15:04:17>,date<17-03-08></p>
CONVERSION	

Functions (The number of executable parameters is specified in square brackets)	General description, example of use
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,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>

Functions (The number of executable parameters is specified in square brackets)	General description, example of use
int[1]	Conversion of fraction into integer (without fractional part). Format: x=int(y); where x—function value, y—argument (fraction for conversion) Example: y=(2.33); x=int(y); Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<2.33>,name<y>,time<16:05:28>,date<22-09-04> Event : CORE VAR_CHANGED int_obj_id<1>,value<2>,name<x>,time<16:05:28>,date<22-09-04>
long2time[1]	It is used to convert specified number of seconds into time. Format: x=long2time(y); where x—function value(time), y—number in seconds Format of the initial recording (argument): <MM> Format of final recording: <HH:MM:SS> Example: x=long2time(12345); Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<03:25:45>,name<x>,time<13:53:02>,date<20-09-04>
time2long[1]	Convert time into a number of seconds. Format: x=time2 long(y); where x—value in seconds, y—time in the <hours>.<minutes> format Example: y=(0.15); x=time2long(y); Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<0.15>,name<y>,time<19:39:49>,date<22-09-04> Event : CORE VAR_CHANGED int_obj_id<1>,value<900>,name<x>,time<19:39:49>,date<22-09-04>
scalar2date[1]	Convert a number of days into a date (number of days is calculated AD). Format: x= scalar2date (y); where x—value(date), y—number of days Example: y=(731500); x=scalar2date(y); Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<731500>,name<y>,time<19:57:46>,date<22-09-04> Event : CORE VAR_CHANGED int_obj_id<1>,value<12-10-03>,name<x>,time<19:57:46>,date<22-09-04>
scalar[1]	Convert date to number of days (number of days is calculated AD). Format: x=scalar(y); where x—numerical value (in days), y—date Recording format: <DD.MM.YYYY> Example: x=scalar("19.10.2004") Event received:

Functions (The number of executable parameters is specified in square brackets)	General description, example of use
	Event : CORE VAR_CHANGED int_obj_id<10>,value<731873>,owner<WS1>,name<x>,time<15:24:11>,guid_pk<{42E93AF5-4862-485E-AEF6-D14C7BF79C5B}>,date<08-12-09>
convert_num[1]	Convert a number into the string. Format: x=convert_num(y); where x—string value of the number, y—convertible number Example: y=(24009921); x=convert_num(y); Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<24009921>,name<y>,time<12:37:20>,date<23-09-04> Event : CORE VAR_CHANGED int_obj_id<1>,value<Twenty four million nine thousand nine hundred twenty-one>,name<x>,time<12:37:20>,date<23-09-04>
convert_cur[1]	Convert a number (sum of money) into the string and add dollars and cents. Format: x=convert_cur(y); where x—string value of the sum of money, y—number (sum of money) Recording format: <DD.CC> Example: y=(17999.98); x=convert_cur(y); Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<17999.98>,name<y>,time<12:49:30>,date<23-09-04> Event : CORE VAR_CHANGED int_obj_id<1>,value<Seventeen thousand nine hundred ninty-nine dollars ninty-eight cents>,name<x>,time<12:49:30>,date<23-09-04>
FORMATTING	
number_frm[2]	Formatting a number. Format: x=number_frm(y,z); where x—function value, y—initial number, z—number of figures after the decimal Example: y=(17999.09998); x=number_frm(y,3); Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<17999.09998>,name<y>,time<14:21:24>,date<23-09-04> Event : CORE VAR_CHANGED int_obj_id<1>,value<17999.100>,name<x>,time<14:21:24>,date<23-09-04>
int_frm[2]	Formatting number. Format: x=int_frm(y,z); where x—value, y- operand, z—number of output digits Example: y=(17999.99); x=int_frm(y,10); Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<17999.99>,name<y>,time<14:31:46>,date<23-09-04> Event : CORE VAR_CHANGED int_obj_id<1>,value<0000017999>,name<x>,time<14:31:46>,date<23-09-04>
currency_std[1]	Formatting currency value (from '.' to '-').

Functions (The number of executable parameters is specified in square brackets)	General description, example of use
	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-F6CF049E58698}>,time<15:02:04>,owner<D-IVANOV> * Id=2 (see value<2>), if the function returns empty value (value<>), then check if the function and its parameters are written correctly
STRING	
strequal[2]	Comparing strings. Format: x= strequal(z,y); where x—value, z and y—compared strings Example: z=str(1019); y=str(1019); x=strequal(z,y); Event received: Event : CORE VAR_CHANGED int_obj_id<1>,value<1019>,name<z>,time<16:51:45>,date<23-09-04> Event : CORE VAR_CHANGED int_obj_id<1>,value<1019>,name<y>,time<16:51:45>,date<23-09-04> Event : CORE VAR_CHANGED int_obj_id<1>,value<1>,name<x>,time<16:51:45>,date<23-09-04> * «value<1>» (see example above)—in the received event we get «value<>»—compared strings differ, or «value<1>»—compared strings are the same
strsub[2]	Determining if there is a substring in the string. Format: x=strsub(y,z); where x—value, y—string in which the search is performed, z—substring Example 1: z=str(888123); y=str(123);

Functions (The number of executable parameters is specified in square brackets)	General description, example of use
	<p>x=substr(z,y);</p> <p>Event received:</p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value<888123>,name<z>,time<16:07:07>,date<23-09-04></p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value<123>,name<y>,time<16:07:07>,date<23-09-04></p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value<4>,name<x>,time<16:04:34>,date<23-09-04></p> <p>Example 2:</p> <p>z="67hb8vc56";</p> <p>y="vc";</p> <p>x=substr(z,y);</p> <p>Event received:</p> <p>Event : CORE VAR_CHANGED value<67hb8vc56>,name<z>,time<12:15:09>,date<18-03-08></p> <p>Event : CORE VAR_CHANGED value<vc>,name<y>,time<12:15:09>,date<18-03-08></p> <p>Event : CORE VAR_CHANGED value<6>,name<x>,time<12:15:09>,date<18-03-08></p> <p>* "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<></p>
strempty[1]	<p>Determining if the string is empty.</p> <p>Format: x=strempty(y); where x—value (1 if string is empty), y—string</p> <p>Example:</p> <p>y="";</p> <p>x=strempty(y);</p> <p>Event received:</p> <p>Event : CORE VAR_CHANGED value<>, name<y>,time<12:27:32>,date<18-03-08></p> <p>Event : CORE VAR_CHANGED value<1>,name<x>,time<12:27:32>,date<18-03-08></p> <p>* function value value <> means that string is not empty</p>
strleft[2]	<p>Left alignment.</p> <p>Format: x=strleft(y,z); where x—aligned string, y—string, z—alignment value</p> <p>Example:</p> <p>y=str(123456789);</p> <p>x=strleft(y,5);</p> <p>Event received:</p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value<123456789>,name<y>,time<18:04:05>,date<23-09-04></p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value<12345>,name<x>,time<18:04:05>,date<23-09-04></p> <p>Note. If z is larger than the number of characters in the string, then the function adds spaces to the initial string on the right until its length becomes z</p>
strmid[3]	<p>Get substring.</p> <p>Format: x=strmid(y,z,w); where x—string value, y—string, z—string position, w—substring length</p> <p>Example:</p> <p>z=(7);//position</p> <p>w=(9);//length</p> <p>x=strmid("get substring (1 - string, 2 - position, 3 - length)",z,w);</p> <p>y=strmid("get substring (1 - string, 2 - position, 3 - length)",17,10);</p>

Functions (The number of executable parameters is specified in square brackets)	General description, example of use
	<p>Event received:</p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value<6>,name<z>,time<14:18:08>,date<24-09-04></p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value<9>,name<w>,time<14:18:08>,date<24-09-04></p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value<substring>,name<x>,time<14:18:08>,date<24-09-04></p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value<1 - string>,name<y>,time<14:18:08>,date<24-09-04></p>
strleft[2]	<p>Get left side of string.</p> <p>Format: y=strleft(s,w); where y—string value, s—string, w—length (from string beginning)</p> <p>Example:</p> <pre>w=(5);//length s("Get left side of string");//string y=strleft(s,w);</pre> <p>Event received:</p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value<5>,name<w>,time<14:54:31>,date<24-09-04></p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value< Get left side of string>,name<s>,time<14:54:31>,date<24-09-04></p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value<Get>,name<y>,time<14:54:31>,date<24-09-04></p>
strright[2]	<p>Get right side of string (1—string, 2—length).</p> <p>Format: y=strright(s,w); where y—string value, s—string, w—length (from string end)</p> <p>Example:</p> <pre>w=(6);// length s("Get right side of string");//string y=strright(s,w);</pre> <p>Event received:</p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value<6>,name<w>,time<15:10:36>,date<24-09-04></p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value< Get right side of string>,name<s>,time<15:10:36>,date<24-09-04></p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value<strings>,name<y>,time<15:10:36>,date<24-09-04></p>
strnleft[2]	<p>Get without left side of string.</p> <p>Format: y=strnleft(s,w); where y—string value, s—string, w—length of left side that will be cut</p> <p>Example:</p> <pre>w=(6);//length s("get without left side of string");//string y=strnleft(s,w);</pre> <p>Event received:</p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value<6>,name<w>,time<15:32:38>,date<24-09-04></p> <p>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></p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value<without left side of string>,name<y>,time<15:32:38>,date<24-09-04></p>
strtright[2]	<p>Get without right side of string.</p> <p>Format: y=strtright(s,w); where y—string value, s—string, w—length of right side that will be cut</p>

Functions (The number of executable parameters is specified in square brackets)	General description, example of use
	<p>Example:</p> <pre>w=(6);//length s("get without right side of string");//string y=strnrigh(s,w);</pre> <p>Event received:</p> <pre>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 of string>,name<y>,time<15:44:31>,date<24-09-04></pre>
<p>get_substr[3]</p>	<p>Get substring (1—string, 2—substring to start with, 3—substring to end with, "\r"—end of string).</p> <p>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)</p> <p>Recording format: <NN.NN></p> <p>Example:</p> <pre>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);</pre> <p>Event received:</p> <pre>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></pre> <p>Example:</p> <pre>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);</pre> <p>Event received:</p> <pre>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></pre>
<p>strltrim[1]</p>	<p>Remove spaces on the left.</p> <p>Format: y=strltrim(w); where y—result string value, w—string</p> <p>Example:</p> <pre>w(" remove spaces on the left");//string y=strltrim(w);</pre> <p>Event received:</p>

Functions (The number of executable parameters is specified in square brackets)	General description, example of use
	<p>Event : CORE VAR_CHANGED int_obj_id<1>,value< remove spaces on the left>,name<w>,time<17:07:49>,date<24-09-04></p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value<remove spaces on the left>,name<y>,time<17:07:49>,date<24-09-04></p>
strrtrim[1]	<p>Remove spaces on the right.</p> <p>Format: y=strrtrim(w); where y—result string value, w—string</p> <p>Example:</p> <pre>w=("Remove spaces on the right ");//string y=strrtrim(w);</pre> <p>Event received:</p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value<Remove spaces on the right>,name<w>,time<17:18:35>,date<24-09-04></p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value<Remove spaces on the right>,name<y>,time<17:18:35>,date<24-09-04></p>
stratrim[1]	<p>Remove spaces on both sides.</p> <p>Format: y=stratrim(w); where y—result string value, w—string</p> <p>Example:</p> <pre>w(" remove spaces on both sides ");//string y=stratrim(w);</pre> <p>Event received:</p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value< remove spaces on both sides>,name<w>,time<17:27:44>,date<24-09-04></p> <p>Event : CORE VAR_CHANGED int_obj_id<1>,value<remove spaces on both sides>,name<y>,time<17:27:44>,date<24-09-04></p>

Note

The date<DD-MM-YY> and time<HH:MM:SS> functions return the current date and time. The pi<3,1415926535897932384626433832795> function returns the value of π .

8.2.6 Examples of scripts in the embedded language

On the page:

- [Example 1.](#)
- [Example 2.](#)
- [Example 3.](#)
- [Example 4.](#)
- [Example 5.](#)
- [Example 6.](#)
- [Example 7.](#)
- [Example 8.](#)
- [Example 9.](#)

- [Example 11.](#)
- [Example 12.](#)

Below one can find some examples of scripts.

8.2.6.1 Example 1.

Display active camera on the analogue monitor.

Implementation:

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

8.2.6.2 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>");
}
```

8.2.6.3 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>");
}
```

8.2.6.4 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;
}

```

8.2.6.5 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+">");
    }
}

```

8.2.6.6 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;
}

```

8.2.6.7 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);

}

}

```

8.2.6.8 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");
    }
}

```

8.2.6.9 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);
}
}

```

8.2.6.10 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;
}

```

8.2.6.11 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 >");
}

```

8.2.6.12 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.


8.3 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 macro 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
Initiated by the operator using the camera context menu (start/stop recording) or by macro	Start recording at CAM 1 REC reaction, stop recording at CAM 1 REC_STOP reaction	The start/stop recording commands 1 and 2 are equal*
Start/stop recording is initiated by the operator using the camera context menu (start/stop recording) or by macro	Start recording at CAM 1 REC priority<0> reaction, stop recording at CAM 1 REC_STOP priority<0> reaction	The stop recording command 1 stops recording started using command 2.
Start/stop recording is initiated by the operator using the camera context menu (start/stop recording) or by macro	Start recording at CAM 1 REC priority<1> reaction, stop recording at CAM 1 REC_STOP priority<1> reaction	The stop recording command 1 stops recording started using command 2.
Start/stop recording is initiated by the operator using the camera context menu (start/stop recording) or by macro	Start recording at CAM 1 REC priority<2> reaction, stop recording at CAM 1 REC_STOP priority<2> reaction	The start/stop recording commands 1 and 2 are equal*
Start/stop recording is initiated by the operator using the camera context menu (start/stop recording) or by macro	Start/stop recording is initiated by the detection tool (for example, 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: <ol style="list-style-type: none"> 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.

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

 * 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.

8.4 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-lray-lrelay"/>
</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 [Administartor'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.

9 The Script object. Programming using the JScript language

9.1 Programming in JScript

- JScript functionality in Intellect
- Programming tools
 - The Script object
 - The Editor-Debugger utility
 - The Debug window
 - Enabling the Debug window
 - Working with Debug window
 - Getting the list of object names, reactions and events in INTELLECT™
- Operations with scripts
 - Creating a script
 - Saving a script
 - Deleting a script
 - Search text in script
 - Replace text in script
- Creating your first script
- Script debugging
 - Script debugging features
 - Creating and using test events
 - Creating test events
 - Running the script with a test event
 - Using debugger windows of the Editor-Debugger utility
 - Displaying messages about starting, verifying, changing and executing scripts in the debugger windows
 - Viewing the script messages
 - Using third-party debugger programs
- Examples of scripts in JScript language

9.2 Appendix 1. Description of the Editor-Debugger utility

- The purpose of the Editor-Debugger utility
- The interface of the Editor-Debugger utility
 - The Editor-Debugger window
 - The Script Debug/Edit tab
 - Description of the Script Debug/Edit tab
 - The Script object panel in the Script Debug/Edit tab
 - The Script Messages tab
 - Description of the Script Messages tab
 - The Script object panel in the Script Messages tab
 - Main menu
 - The Filter dialog window
 - The Color dialog window
 - The toolbar of the Editor-Debugger utility

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

- Purpose of virtual objects and their implementation in Intellect
- How to create a virtual object
 - DBI file preparation
 - DDI file preparation
 - XML file preparation
 - Creating and using a virtual object in Intellect

9.4 Programming Guide (JScript). Conclusion

9.5 Programming in JScript

9.5.1 JScript functionality in Intellect

The JScript programming language is used in Intellect to implement additional user functions not included in the basic Intellect functionality.

JScript is a de-facto standard for developing and running user scripts. Intellect supports the version of JScript based on ActiveX technology by Microsoft. The general description of the JScript object model is given in the Microsoft documentation (for example, MSDN).

The JScript scripts in Intellect are executed using the standard ActiveX modules included in the Windows operating system. So, any objects from the ActiveX-based JScript can be used in developing the scripts for Intellect.

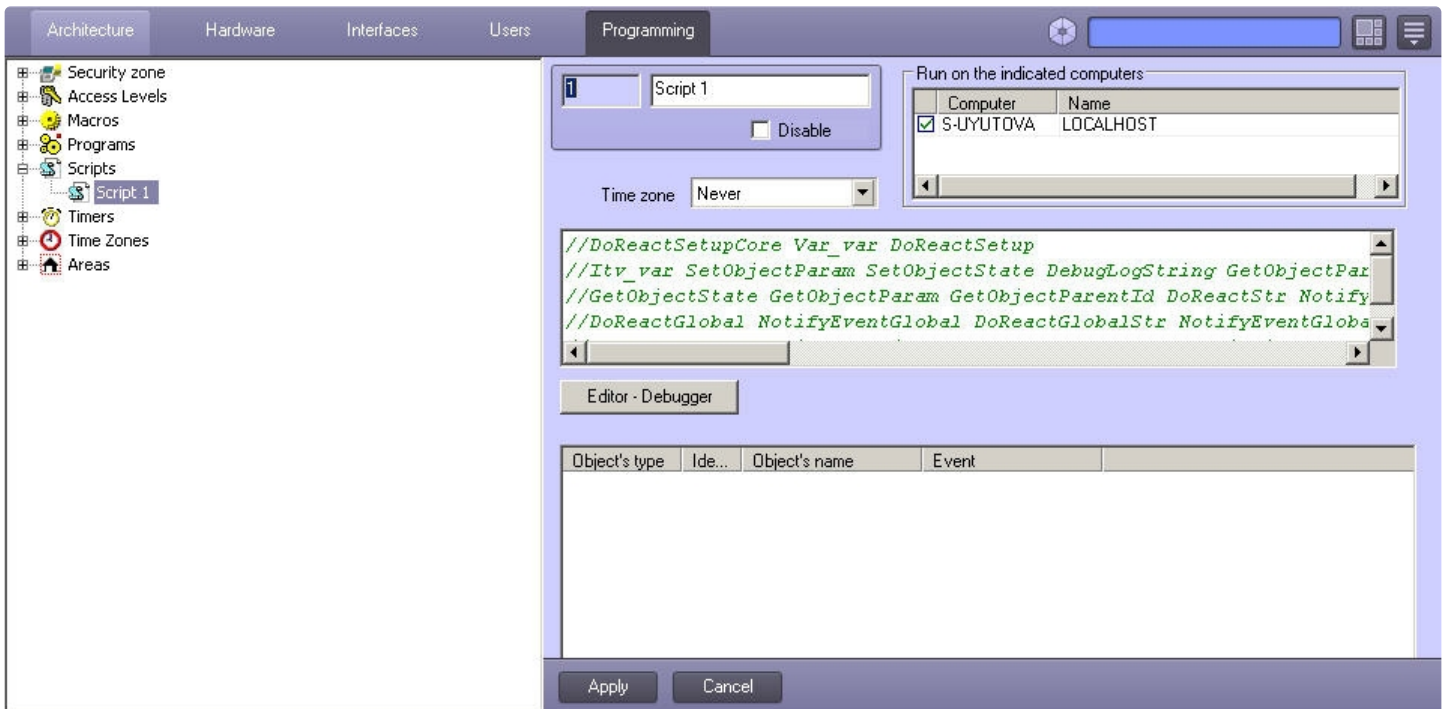
A set of specialized JScript objects is provided in Intellect for handling Intellect system objects, and for sending and receiving system events and actions.

9.5.2 Programming tools

9.5.2.1 The Script object

The **Script** object is designed for initializing and setting the parameters of JScript scripts in Intellect system.

Figure shows the settings panel for the **Script** object.



⚠ Attention!

Creating of large number of the **Script** objects (more than 100) can result to instable system working.

The settings panel allows choosing the time zone and the computers (kernels) for executing the script.

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.

It also has the button for starting the *Editor-Debugger* tool and the text window for viewing the script text created by means of this tool. The scripts can be edited in the *Editor-Debugger* tool or directly in the settings panel for the **Script** object.

Moreover, one can configure the filter of events – the list of events which are to be processed by the **Script** system object. Including the event to the filter equals to *if* operator in the text of script, i.e. when the event is in the table, the operator can be omitted.

Attention!

The event filter is to be set when creating a script in large distributed configurations. Otherwise the module will process all incoming events and it will lead to module malfunctioning.

Example

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

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

script can be used instead of script.

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

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

Example.

Recording on Camera 1 is to be started when controlling PTZ device in the Video surveillance monitor.

For this adjust the **Script** object as follows:

1. Select the required time zone when the script is to be executed.

2. Enter the script text:

1 Script 1 Disable

Time schedule Always

Run on specified computers

Computer	Name
<input checked="" type="checkbox"/> S-UYUTOVA	LOCALHOST

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

Editor - Debugger

Object's type	Ide...	Object's name	Event
CORE			DO_REACT

Apply Cancel

3. Adjust the filter as follows:

1. Select CORE in the **Object's type** dropdown list.
2. Enter DO_REACT in the **Event** field.

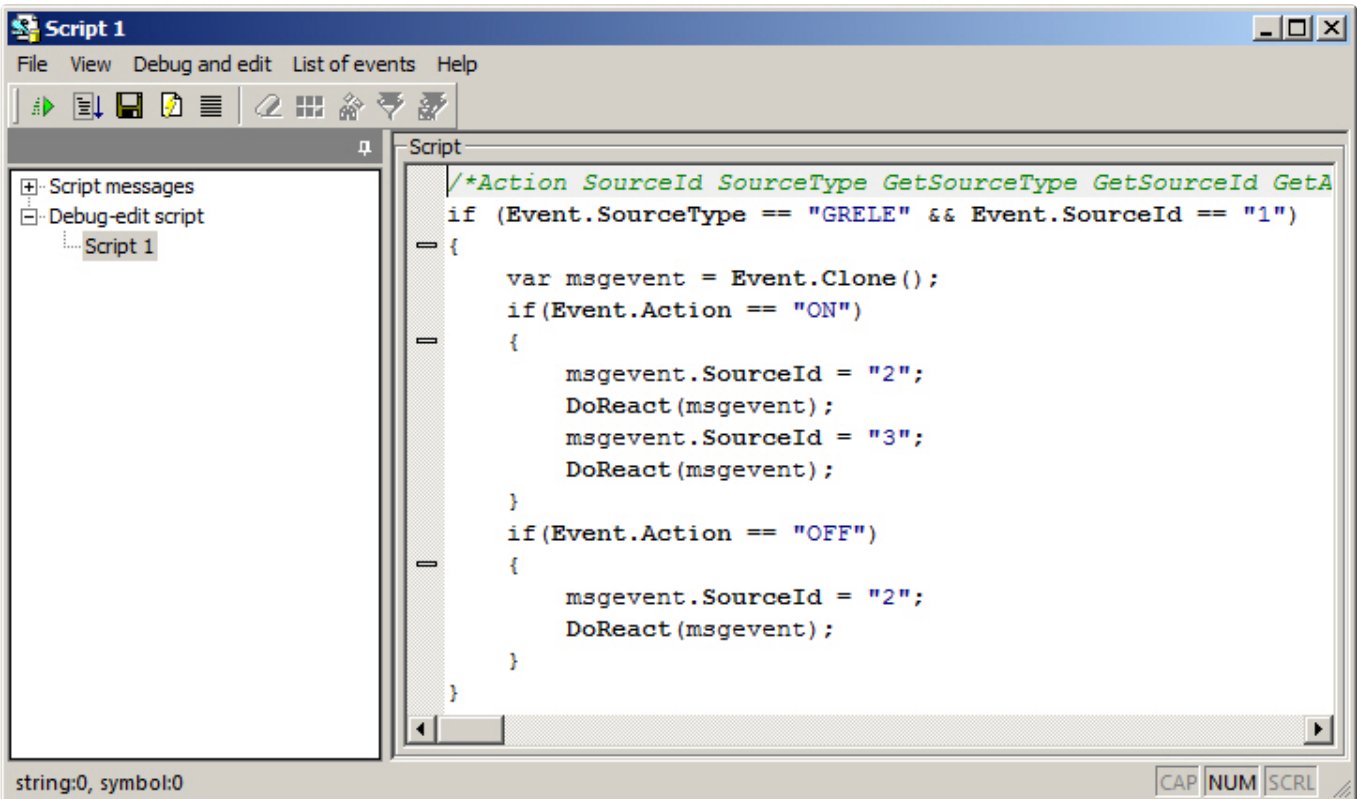
The filter can be set by the UPDATE_OBJECT event of the CORE object. Example command to add **Camera 1** object to the **Script 2** filter:

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

9.5.2.2 The Editor-Debugger utility

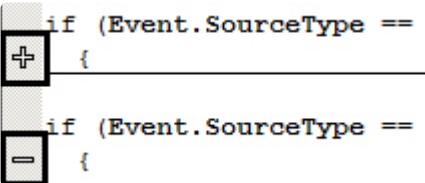
The *Editor-Debugger* utility is designed for creating, debugging and editing scripts in the Intellect software package.

Figure shows the *Editor-Debugger* dialog window.



The *Editor-Debugger* utility contains the built-in text editor and the debugger window.

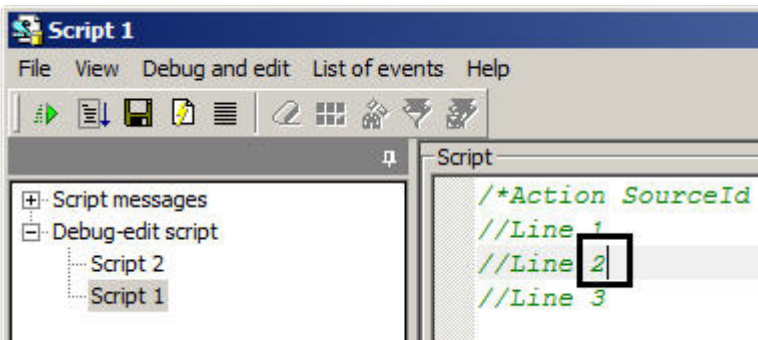
To help with writing correct codes, the text editor automatically highlights objects, methods and properties in different colors. In addition, the code blocks can be collapsed or expanded with - or + buttons in the left of the **Script** text field.



Note.

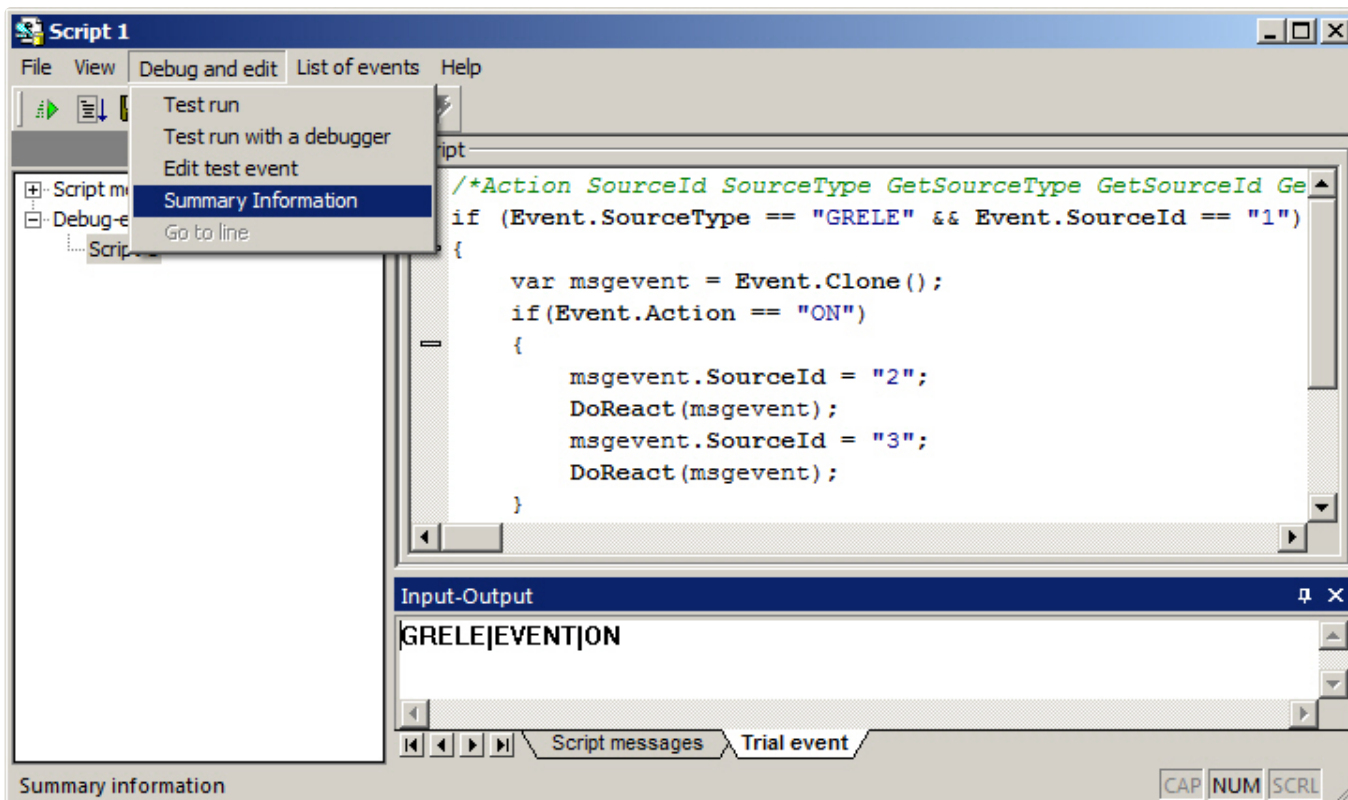
When switching between scripts, script messages or other objects in the *Editor-Debugger* utility tree, the collapsing settings are reset, i.e. all blocks in the script become expanded.

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



The debugger window allows viewing the information about all events registered by the system. You may filter the events to be shown in the debugger window. A separate debugger window is created for each **Script** object, which allows each script to be debugged individually using the filters.

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



Scripts can be saved as **Script** objects or as text files on the hard drive.

9.5.2.3 The Debug window

Intellect allows viewing all events and reactions happening in the system in real time mode. Events and reactions with object properties are displayed in the **Debug window**. They can be copied to the Windows clipboard and then used in programs.

9.5.2.3.1 Enabling the Debug window

By default, the **Debug window** is disabled. To enable the **Debug window**, do the following:

1. Shut down *Intellect*.
2. Run the *Tweak.exe* utility.

Note


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

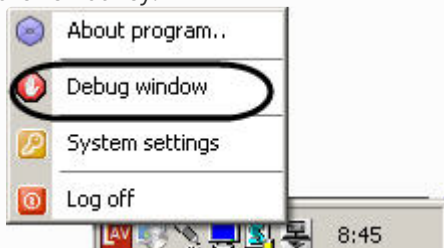
3. Select **Intellect** section in the tree on the left side of the utility dialog box.
4. Change the value of the **Debug mode** parameter from **Disabled** to **Debug 1**, **Debug 2**, **Debug 3**, or **Debug 4**. Any of these modes will enable the **Debug window**, the difference between them is in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

7. A new **Debug window** item will appear on the Main control panel of *Intellect*.

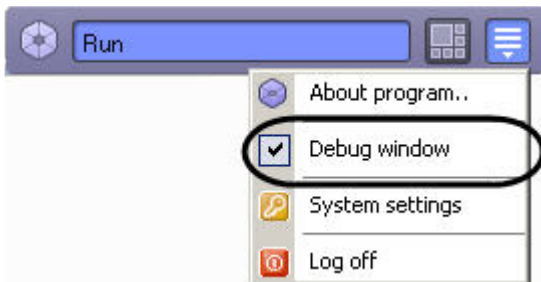


Note

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



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



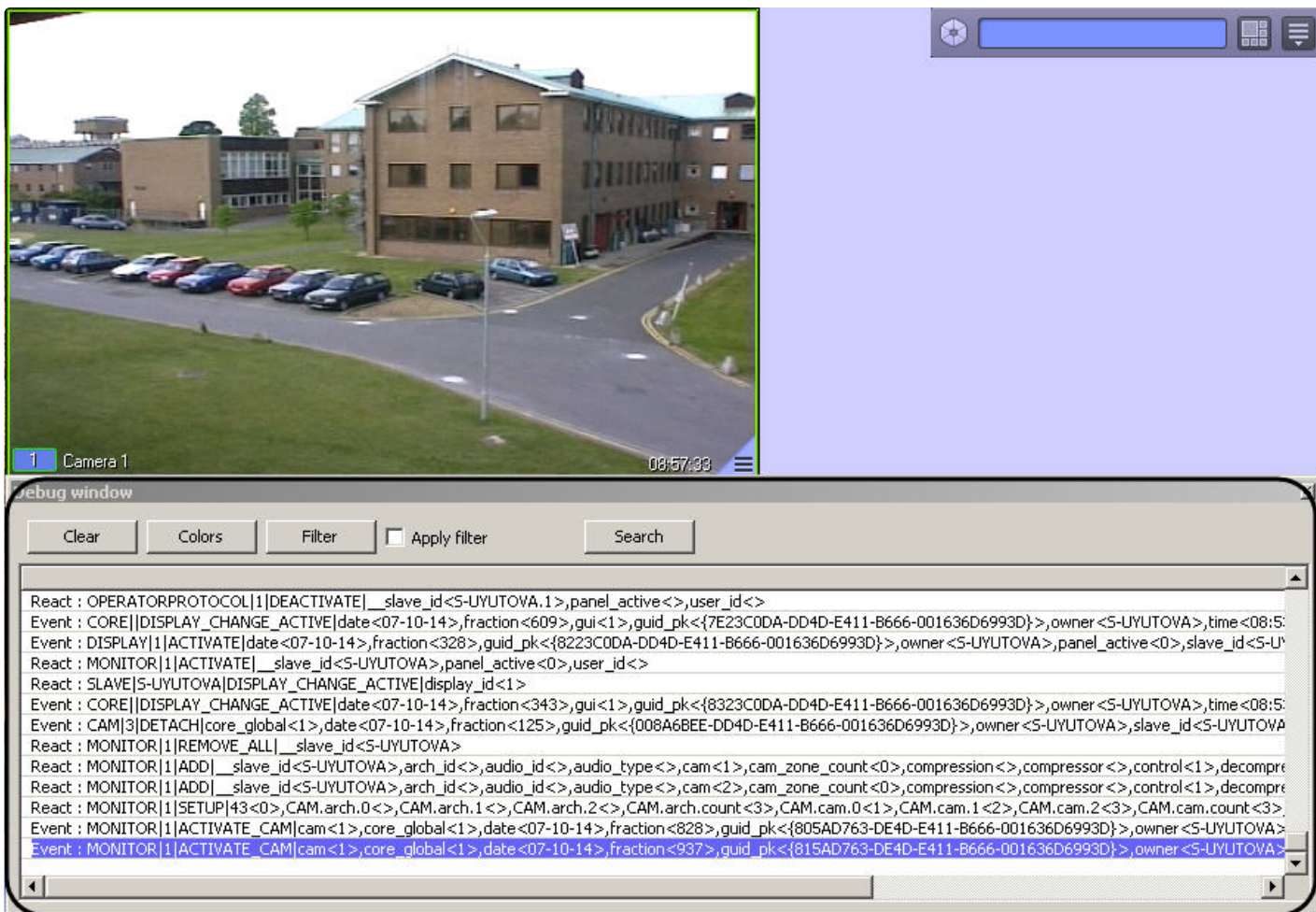
To hide the Debug window, re-select the Debug window item on the Main control panel.

Note

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

9.5.2.3.2 Working with Debug window

Look at the figure of the Debug window. The Debug window displays the sequence of events and reactions in the system.



Here are the features of the Debug window:

1. always on top;
2. the size of the Debug window is changed using the mouse;
3. information on event or reaction can be copied to the Windows clipboard and then used in programs;
4. events and reactions can be filtered in the Debug window;
5. events and reactions can be highlighted in the Debug window;
6. it is possible to search for events or reactions in the Debug window.

Regular expressions can be used in order to highlight and filter messages in the Debug window.

9.5.2.3.2.1 Copying information on event or reaction to the clipboard

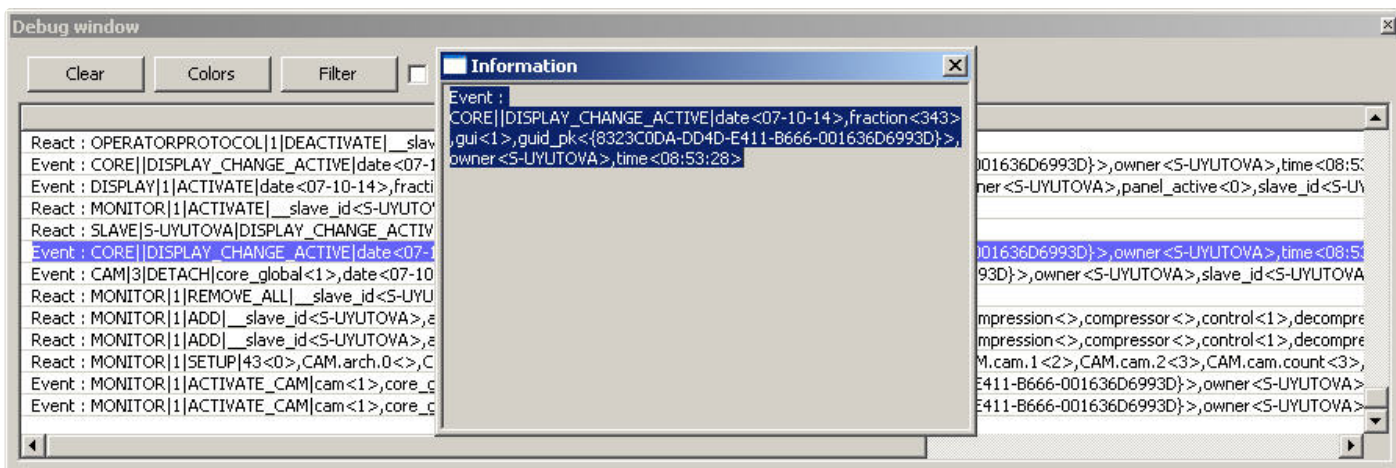
To read and/or copy data on event or reaction to the Windows clipboard, do the following:

1. Highlight the required line in the **Debug window**.
2. Right-click on the highlighted line. The **Info** window with info on required event or reaction appears.
3. Highlight the information that is to be copied to the Windows clipboard and click **Ctrl+C**.

Note.

Use the context menu for operations with text in the **Info** window (right-click on the highlighted text).

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

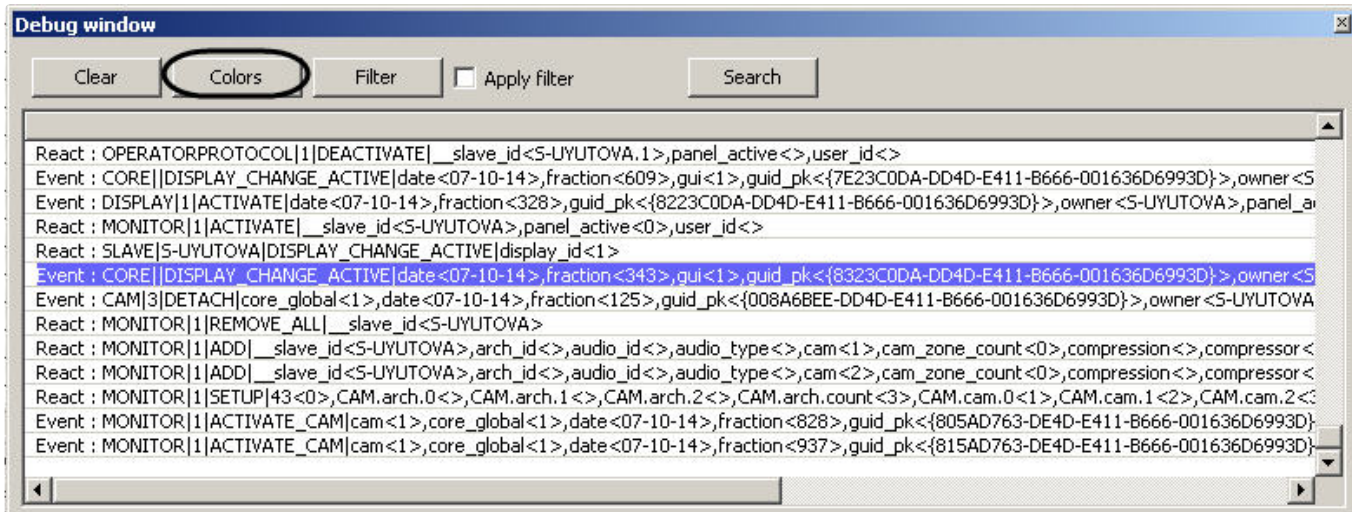


Information on event or reaction is now copied to the Windows clipboard.

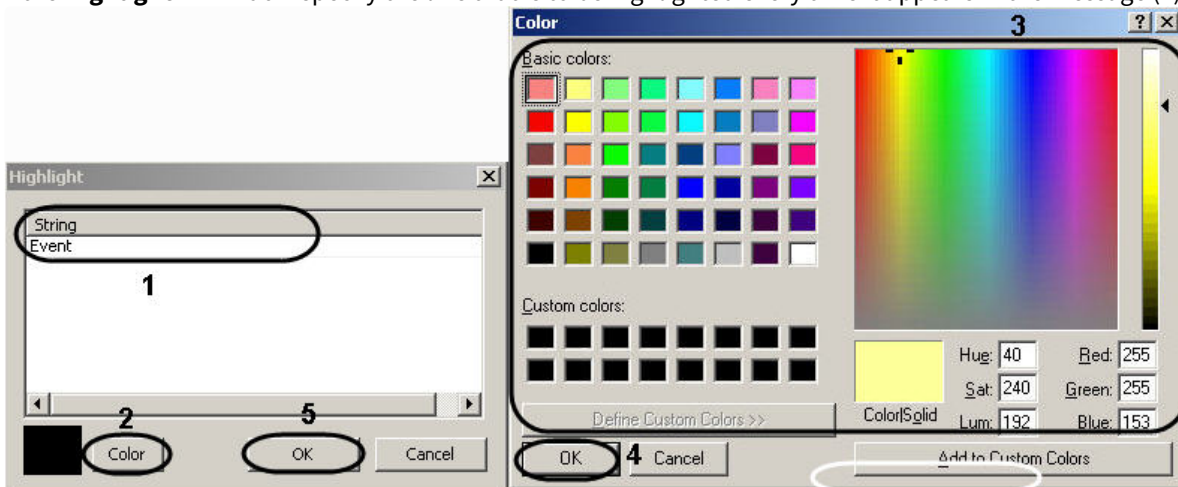
9.5.2.3.2.2 Highlighting messages

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

1. Click the **Colors** button.



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



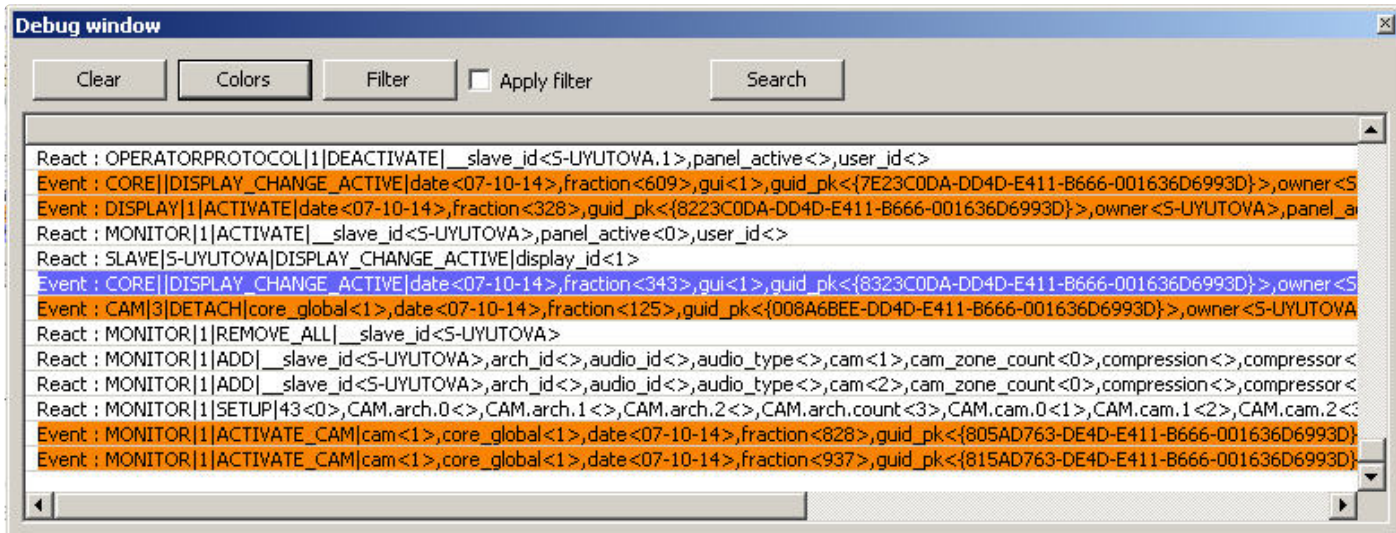
3. Click the **Color** button (2).
4. Select the color in the **Color** common dialog box (3).
5. Click the **OK** button(4).
6. Repeat steps 2-5 for all required lines.

Note.

To add a line to the table, click the ↓ key on the keyboard.

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

As a result messages with the specified line are highlighted in the Debug window.



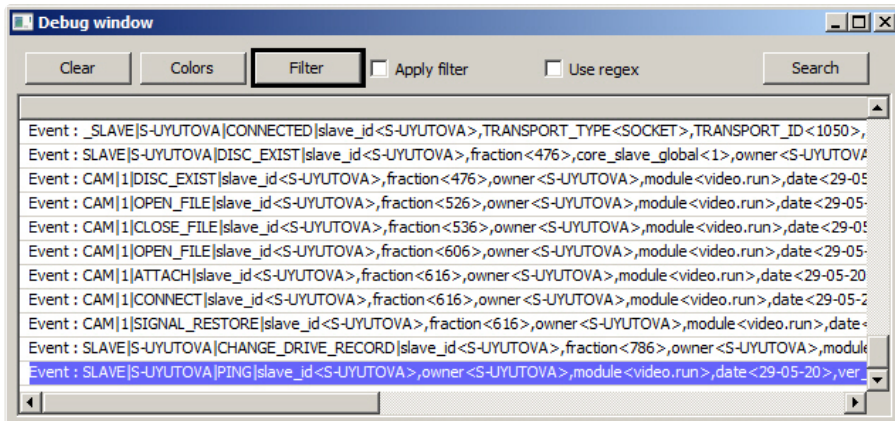
The messages are now highlighted.

9.5.2.3.2.3 Event and reaction filter

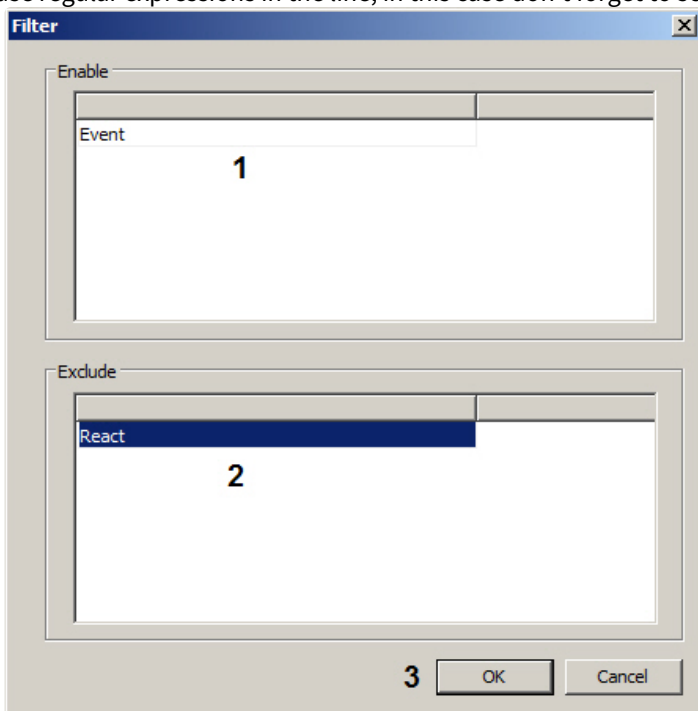
The filter of events and reactions allows displaying only required messages in the Debug window.

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

1. Click the **Filter** button.



- In the **Filter** window specify the lines that are to be in the message for it to be displayed in the Debug window (1). You can use regular expressions in the line, in this case don't forget to set the **Use regex** checkbox on step 5.

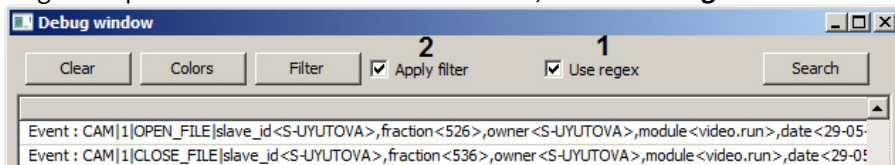


- Specify the lines that are to be in the message for it to not be displayed in the Debug window (2). You can use regular expressions in the line, in this case don't forget to set the **Use regex** checkbox on step 5.

Note.

To add a line to the table, click the ↓ key on the keyboard.

- Click the **Yes** button (3).
- If regular expressions were used in the filter lines, set the **Use regex** checkbox checked (1).



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

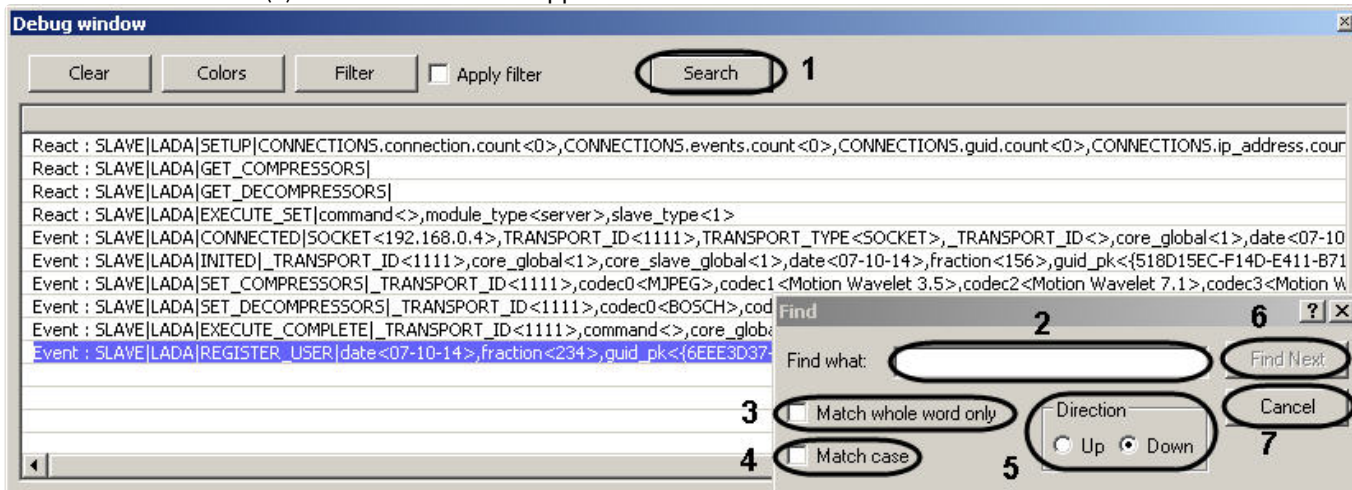
As a result only messages that fit the search criteria are displayed in the Debug window.

The event and reaction filter is now configured.

9.5.2.3.2.4 Searching for events and reactions

To search for events and reactions, do the following:

1. Click the **Search** button (1). The **Search** window appears.



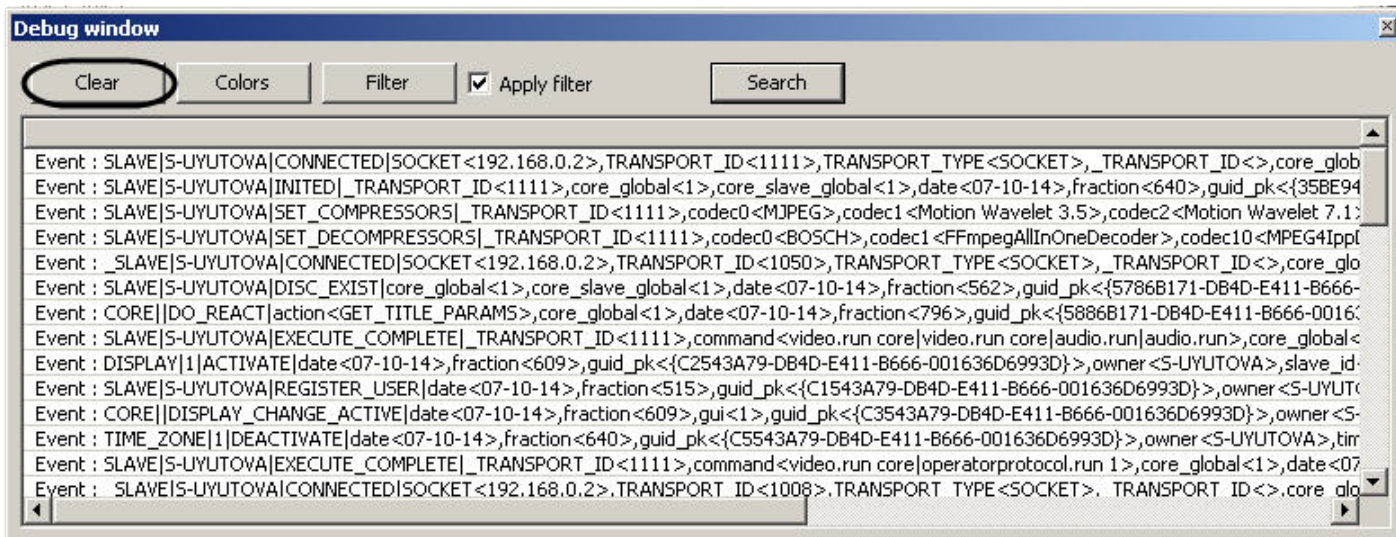
2. Specify the search criteria in the **Search for:** field (2).
3. If the specified string is to be searched as a whole word, not a part of any other word, then set the **Match whole word** checkbox checked (3).
4. If the search is to be case sensitive, then set the **Case sensitive** checkbox checked (4).
5. Set the **Direction** switch into the position corresponding to the search direction (5).
6. To view the following search result, click the **Find next** button (6).

Note.
To close the **Search** window click the **Cancel** button.

The search for events and reactions is now complete.

9.5.2.3.2.5 Clearing the Debug window

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



9.5.2.4 Getting the list of object names, reactions and events in INTELLECT™

The list of object names, reactions and events used while programming in *INTELLECT™* software can be got with the help of ddi.exe tool. See the description of the main system objects reactions in the [Description of events and reactions of system objects](#).

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

1. “**Start** → **All Programs** → **Intellect** → **Tools** → **System configuration**.”

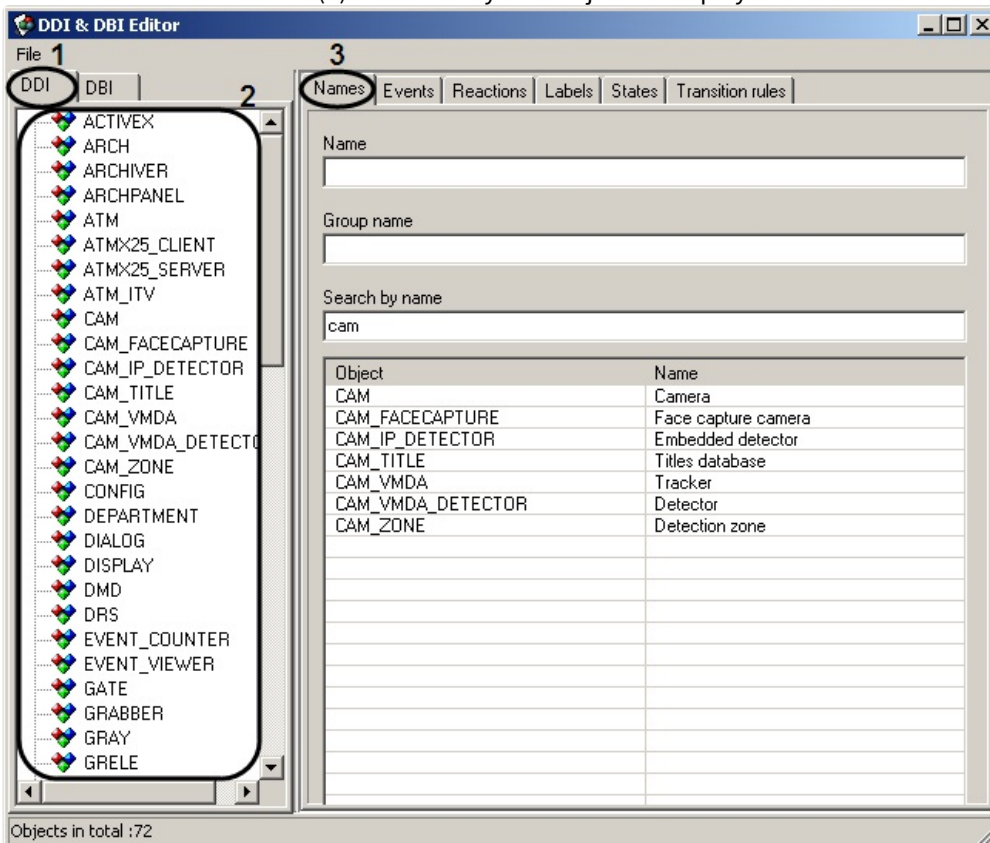
Note

The *ddi.exe* tool is available in the **Start** menu with the following types of Intellect installation: Server, Administrator’s workstation and Workstation for monitoring

2. In the **Tools** folder of Intellect installation folder.

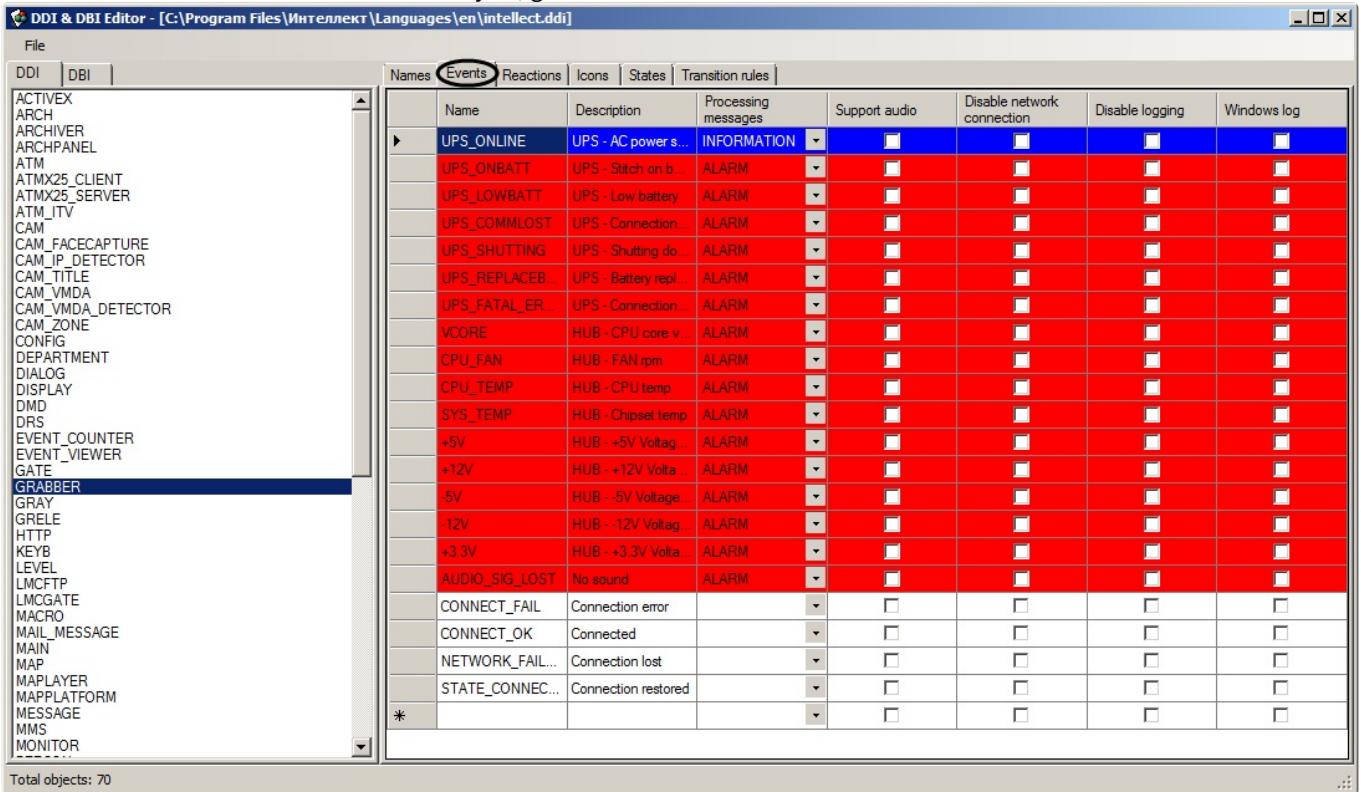
To view the list of object names, events and reactions, do the following:

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

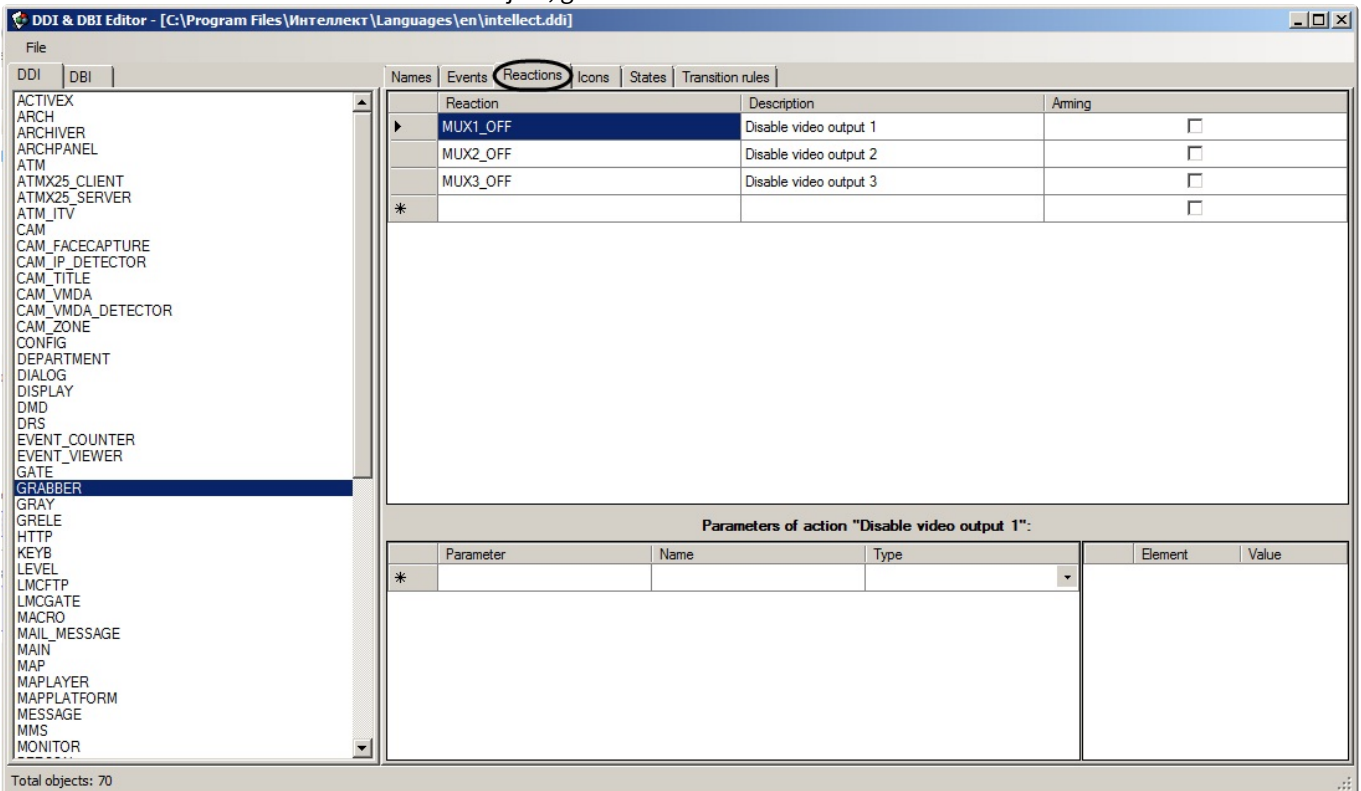


3. In the **DDI** tab, select the object which events and reactions are to be viewed (2).
4. To view the name of the selected object, go to the **Names** tab (3).

5. To view the list of events for the selected object, go to the **Events** tab.



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



See the detailed information on working with ddi.exe tool in the [Editing intellect.dbi and intellect.ext.dbi database templates using the ddi.exe utility.](#)

Note

If the Sensor is armed, then at the Sensor switching on/off the "Alarm" event appears depending on the alarm mode setting (see the [Creating and configuring the Sensor system object](#) section of the [Installing and configuring security system components guide](#)). If the Sensor is disarmed, the "Closed" / "Opened" events appear correspondingly.

9.5.3 Operations with scripts

9.5.3.1 Creating a script

On the page:

- [Creating the Script object](#)
- [Creating and editing a script](#)
 - [Features when working with a script](#)
- [Debugging a script](#)

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

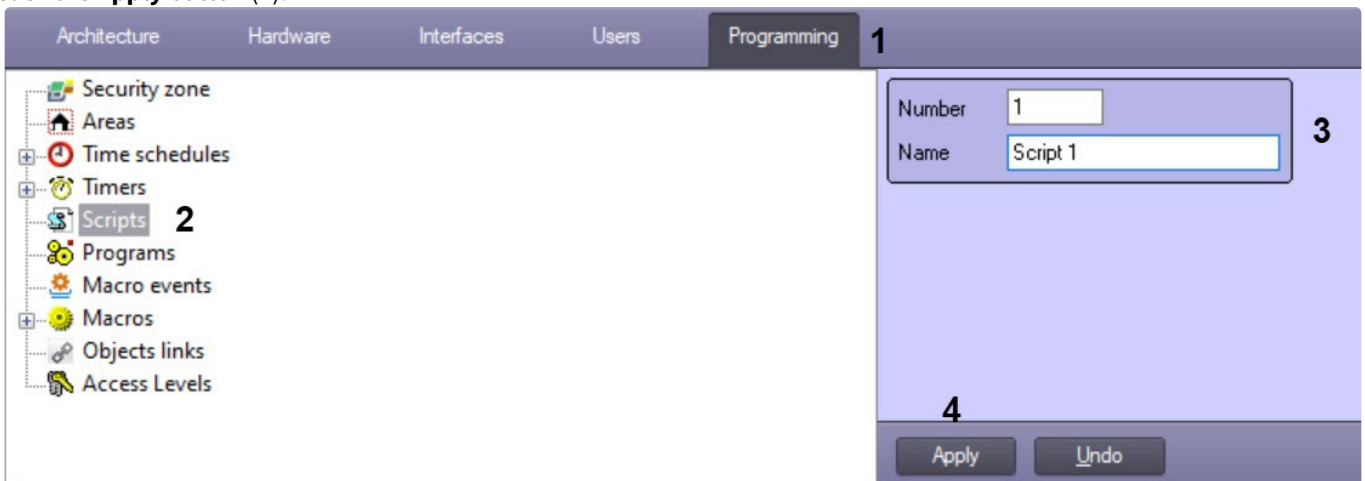
You can create JScript scripts in *Intellect* using the built-in *Editor-Debugger* utility.

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

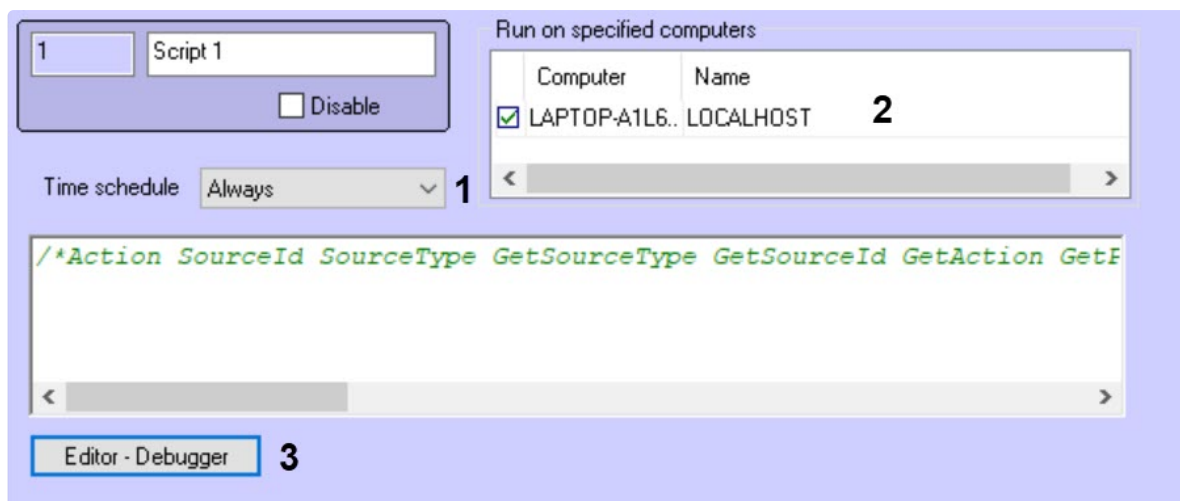
9.5.3.1.1 Creating the Script object

To create the **Script** object in *Intellect*, do the following:

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



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



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

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

Note

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

3. Click the **Apply** button.

9.5.3.1.2 Creating and editing a script

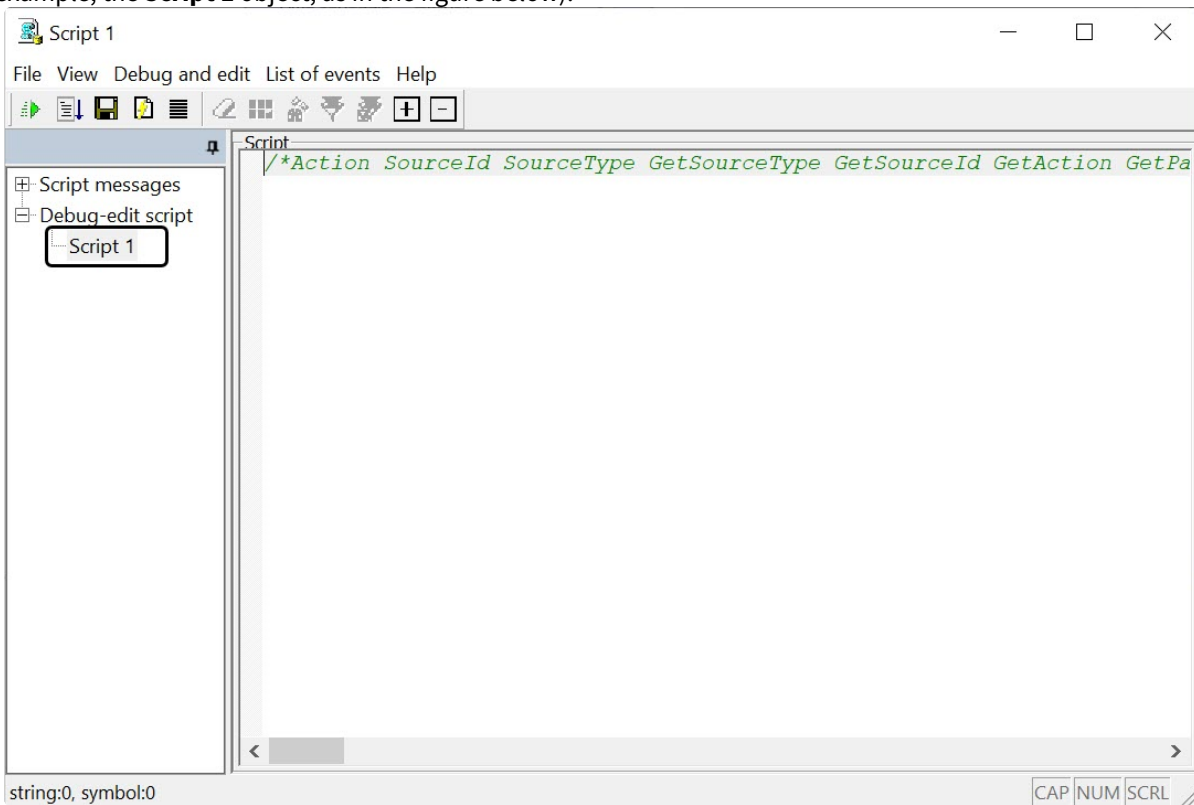
To create a script in JScript in *Intellect*, do the following:

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

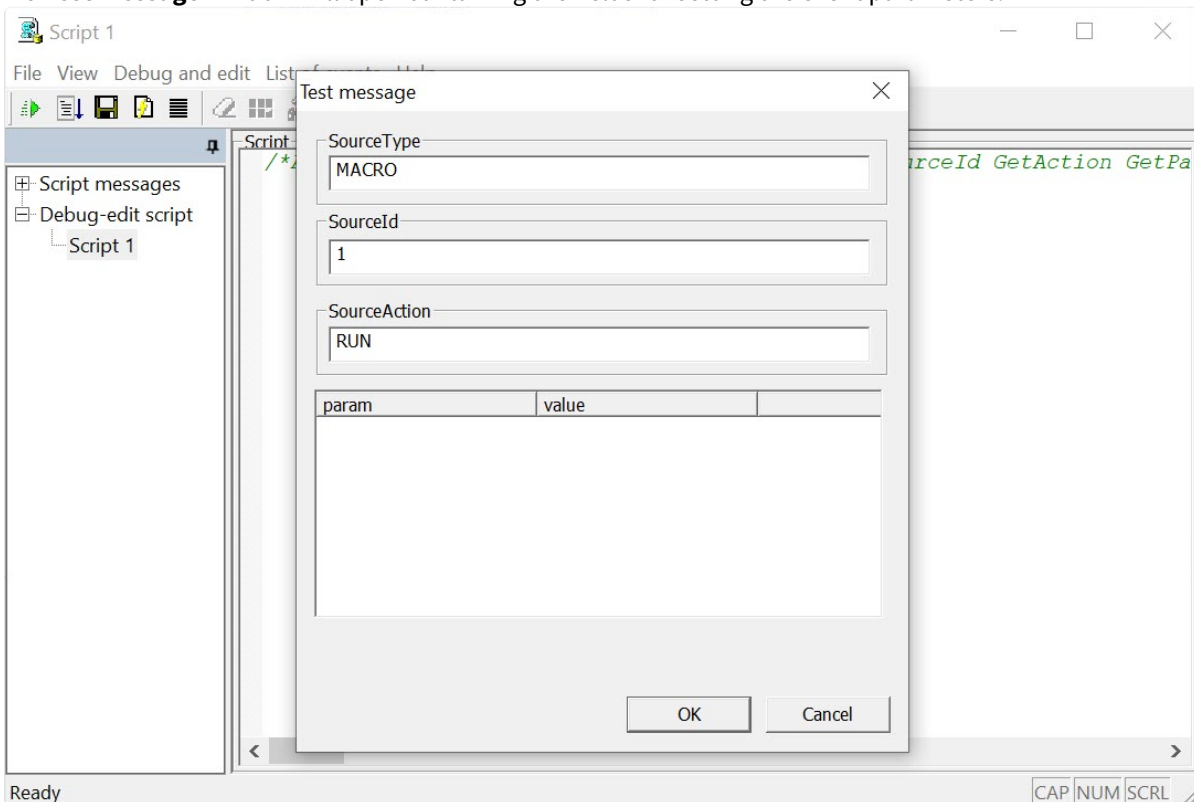
Note

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

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



- In the **Script** field, enter the text of the script in JScript programming language (see [Examples of scripts in JScript language](#)).
- Run the script using a test event. To create a test event, select **Debug and edit** → **Edit test event**. The **Test message** window will open containing the fields for setting the event parameters.



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

Note

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

9.5.3.1.2.1 Features when working with a script

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

Note

For your convenience, when editing the script, the cursor position is saved when saving the script or when switching between the scripts in the *Editor-Debugger* utility window during a session, i.e. until *Intellect* is restarted.

Note

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

9.5.3.1.3 Debugging a script

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

Note

See the detailed information about using test events for script debugging in [Script debugging](#).

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

Script creation is considered complete if it runs correctly.

9.5.3.2 Saving a script

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

To save the script in the **Script** object, in the **File** menu, select **Save in the database**. When this item is selected, the script is updated on every Server selected on the **Script** object settings panel (see [The Script object](#)).

Note.

Also, use **Ctrl+S** hotkey to save the script in the database.

Note

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

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

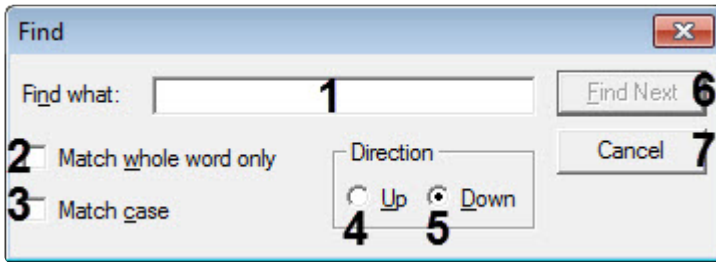
9.5.3.3 Deleting a script

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

9.5.3.4 Search text in script

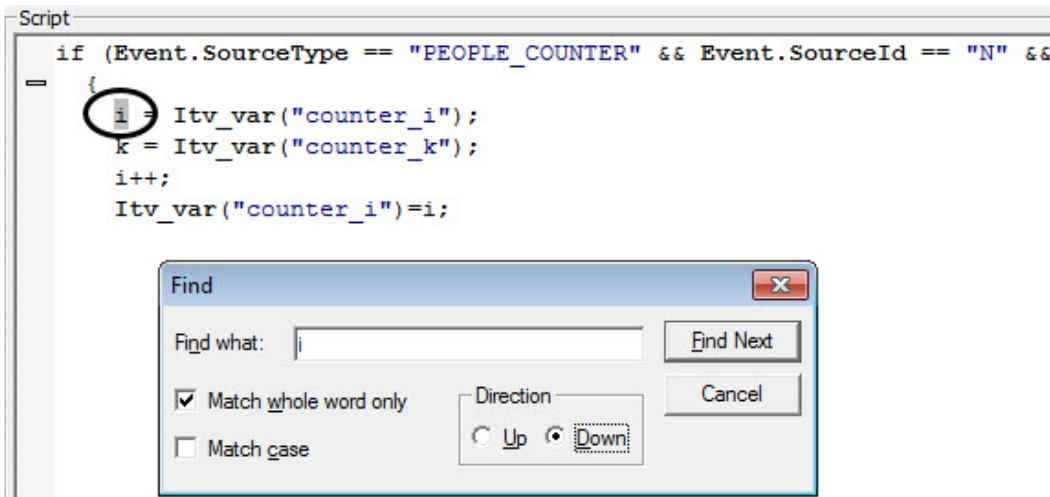
The *Editor-Debugger* allows searching text in script using a dialog box.

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



1. Enter the text to find in the **Find what:** field (1).
2. Set the **Match whole word only** checkbox to search for the text in whole (2).
3. Set the **Match case** checkbox to case-sensitive search for the text (3).
4. Select the direction of search through the text relatively to the current cursor position: **Up** (4) or **Down** (5).
5. Click **Find Next** to run search or go to the next entry (6).
6. Click **Cancel** to stop search and close the dialog box.

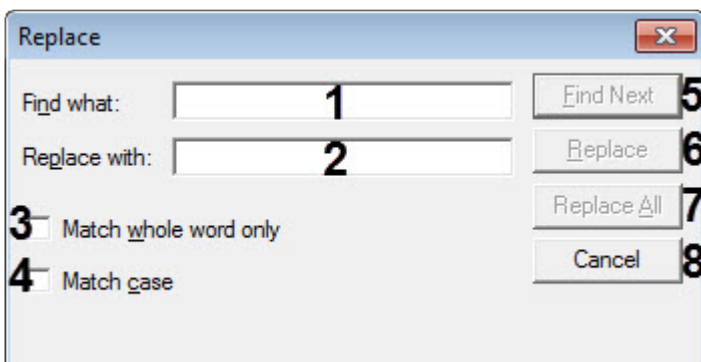
The found match is highlighted in the *Editor-Debugger* utility window.



9.5.3.5 Replace text in script

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

Press **Ctrl+H** to open replace dialog box. The **Replace** dialog box opens.



1. Enter the text to find in the **Find what:** field (1).
2. Enter the text to replace the found text with in the **Replace with:** field (2).

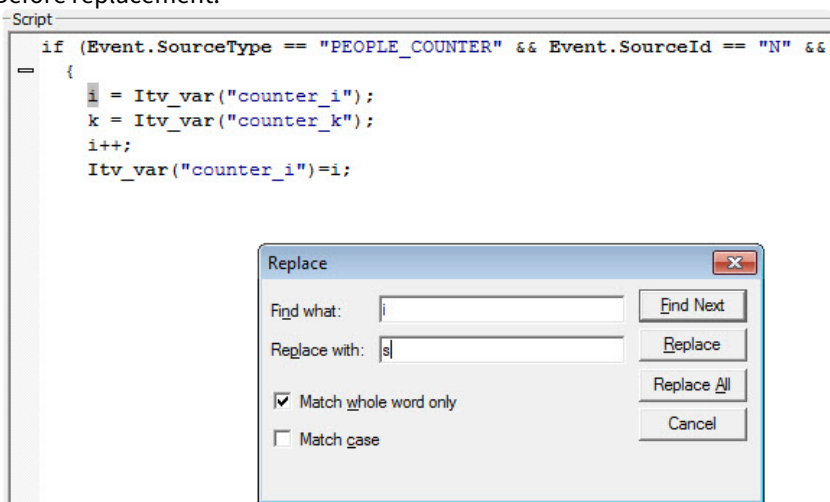
3. Set the **Match whole word only** checkbox to search for the text in whole (3).
4. Set the **Match case** checkbox to case-sensitive search for the text (4).
5. Click **Find Next** to run search or go to the next entry (5).

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

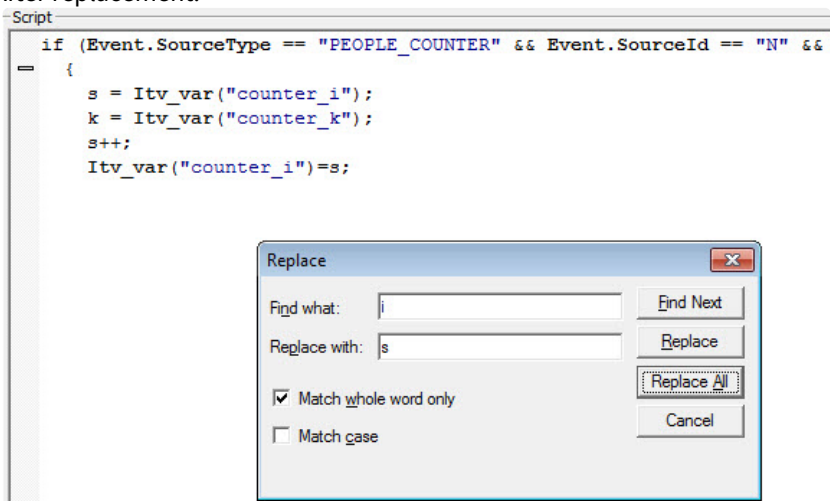
6. Click **Replace** to replace the current found match (6).
7. Click **Replace All** to replace all matches automatically (7).
8. Click **Cancel** to stop close the replace dialog box (8).

Example of use fo replacing variable **i** to **s**:

1. Before replacement:



2. After replacement:



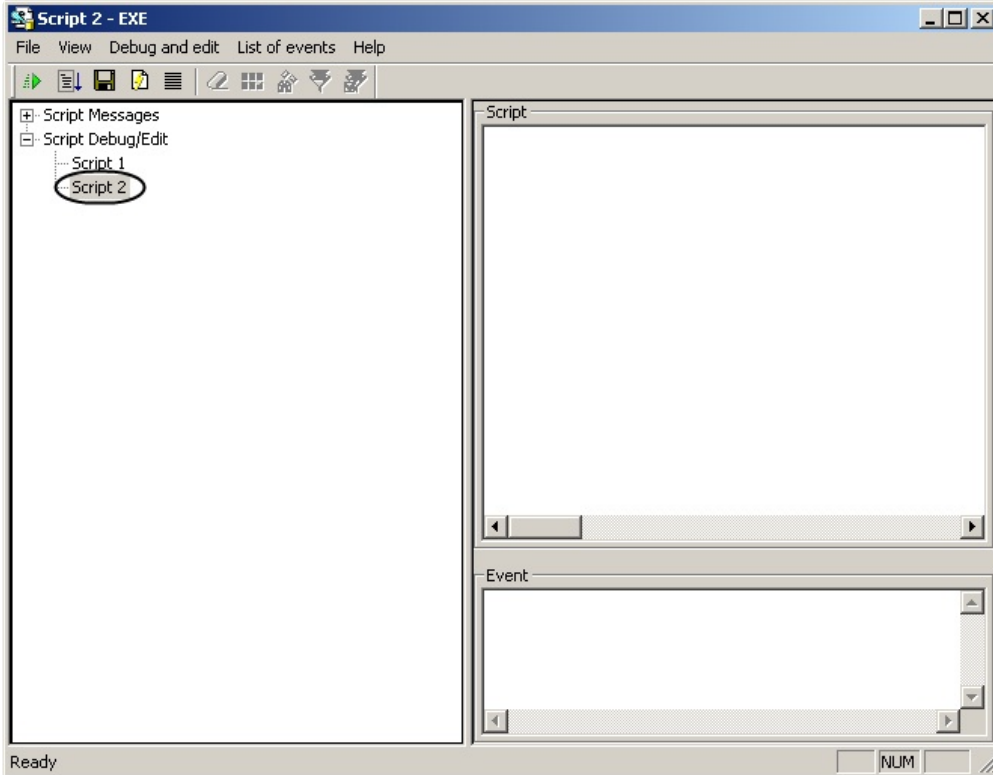
9.5.4 Creating your first script

As an example of using JScript in *Intellect*, try to create a script for the following tasks: when Macro 1 starts, set the value 10 to the **Hot Recording** parameter for Cameras № 1 – 4 and output the **Hello world** message to the debugger window of the *Editor-Debugger* utility.

To create and run this script, do the following:

1. In the **Hardware** tab of the **System Settings** window, create four **Camera** objects with identification numbers 1, 2, 3 and 4, if they have not been created before.
2. In the **Programming** tab, create a **Macro** object with identification number 1. The Events table should not be filled for .

3. Create a **Script** object in the **Programming** tab. Enter the identification number 1 and the name **Script 1**.
4. In the **Script 1** object settings panel, select **Always** in the Time zone list.
5. Click the **Editor-Debugger** button at the bottom of the **Script 1** settings panel. The *Editor-Debugger* window will open.
6. In the *Editor-Debugger* window, open the **Script Debug/Edit** list and select the **Script 2** object.



7. Enter the following in the **Script** field:

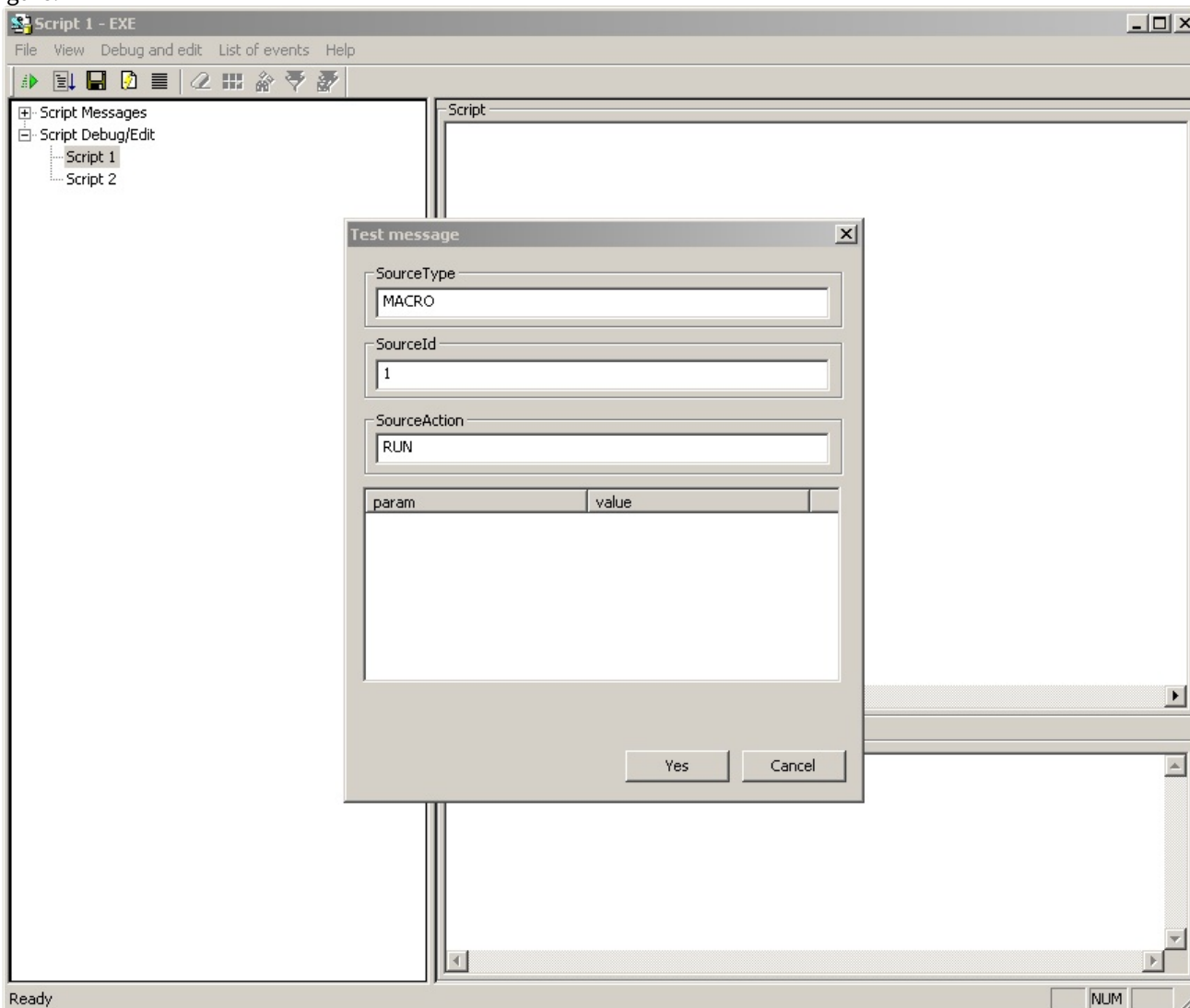
```

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

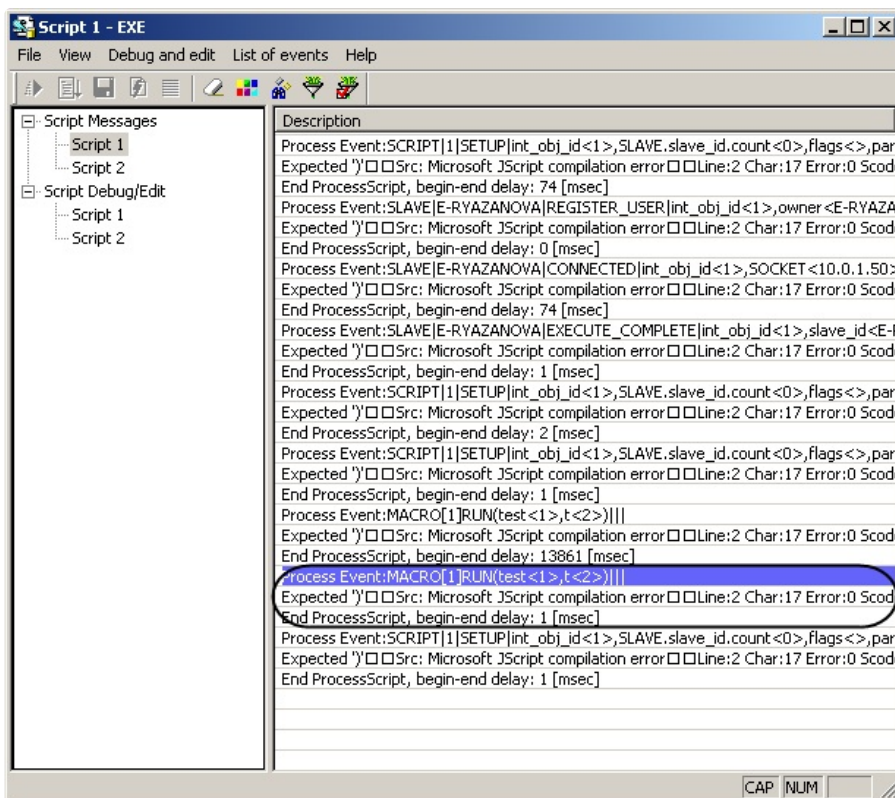
```

8. In the **File** menu, select **Save to database** to save the script.
9. Create a test event to run the script in debug mode – “MACRO|1|RUN|”. To achieve this, in the **Debug and edit** menu, select **Edit test event**; the **Test message** window will open. Fill in the fields in the **Test message** window as shown in

figure.



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



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

- Change the text of the script (see the “var i;” line).

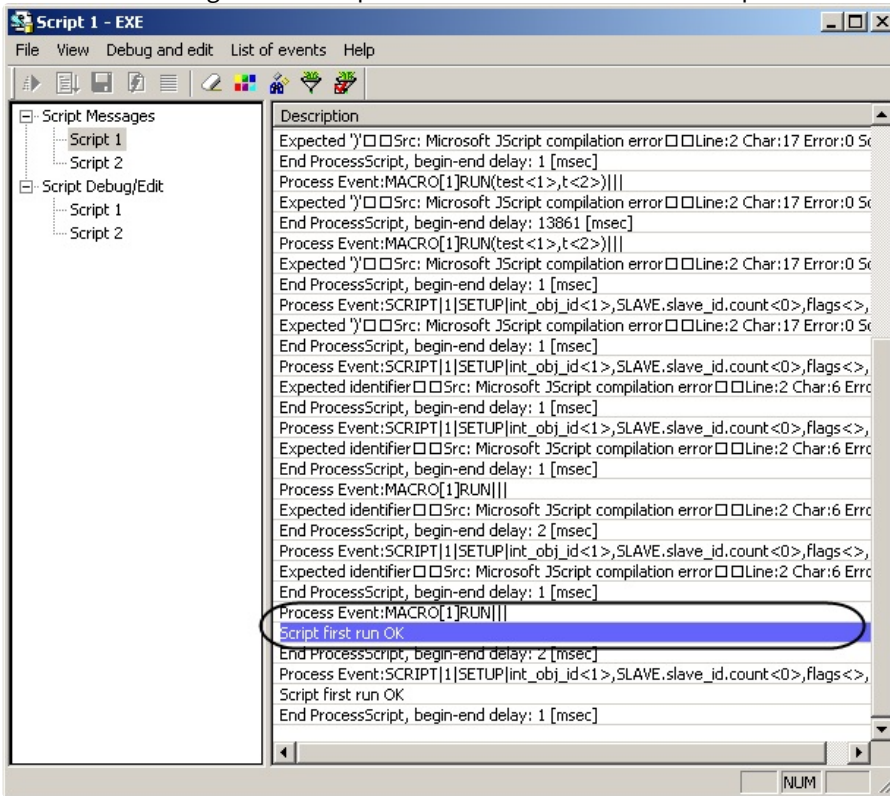
```

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

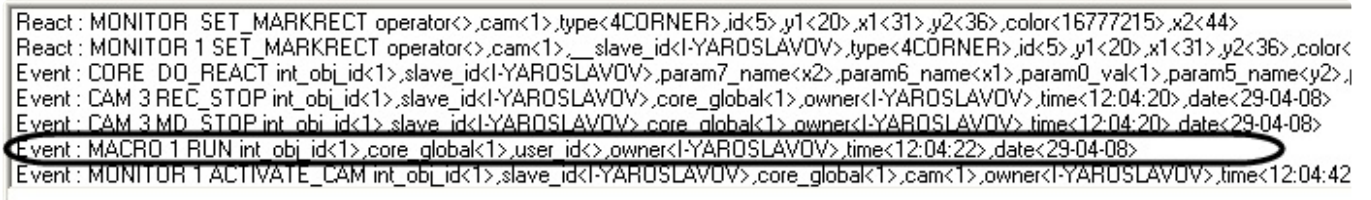
```

- In the **File** menu, select **Save to database** to save the script.
- Repeat steps 10 and 11.

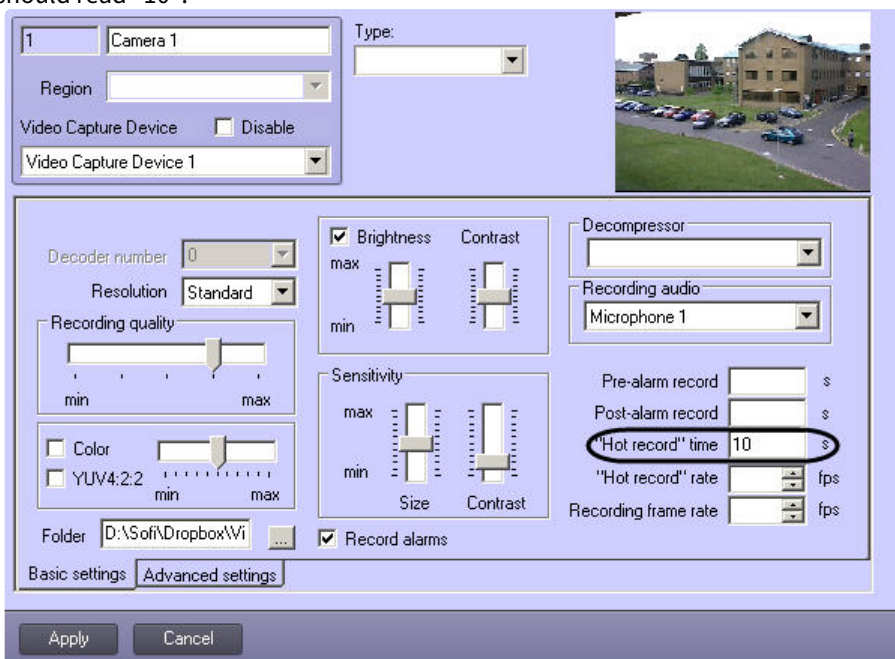
- In the debugger window, find the “Process Event:MACRO|1|RUN|” line and the “DebugLogString:Hello world” and “Script first run OK” messages. The “Script first run OK” means that the script runs correctly in the debug mode.



- Close the *Editor-Debugger* utility.
- The text of the created script will be shown in the **Script** field of the **Script 1** object settings panel. Click the **Apply** button in the **Script 1** settings panel to activate the script.
- Select **Macro 1** in the **Run** menu of the main control panel.
- In the debugger window of the *Intellect* system, check that the macro and the script have run successfully.



21. Check the accuracy of the script result. The **Hot recording field** in the **Camera 1** to **Camera 4** object settings panels should read "10".



Note
The **Hot recording field** in the **Camera** settings panel is empty by default

Script creation and debugging is now complete.

9.5.5 Script debugging

9.5.5.1 Script debugging features

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

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

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

9.5.5.2 Creating and using test events

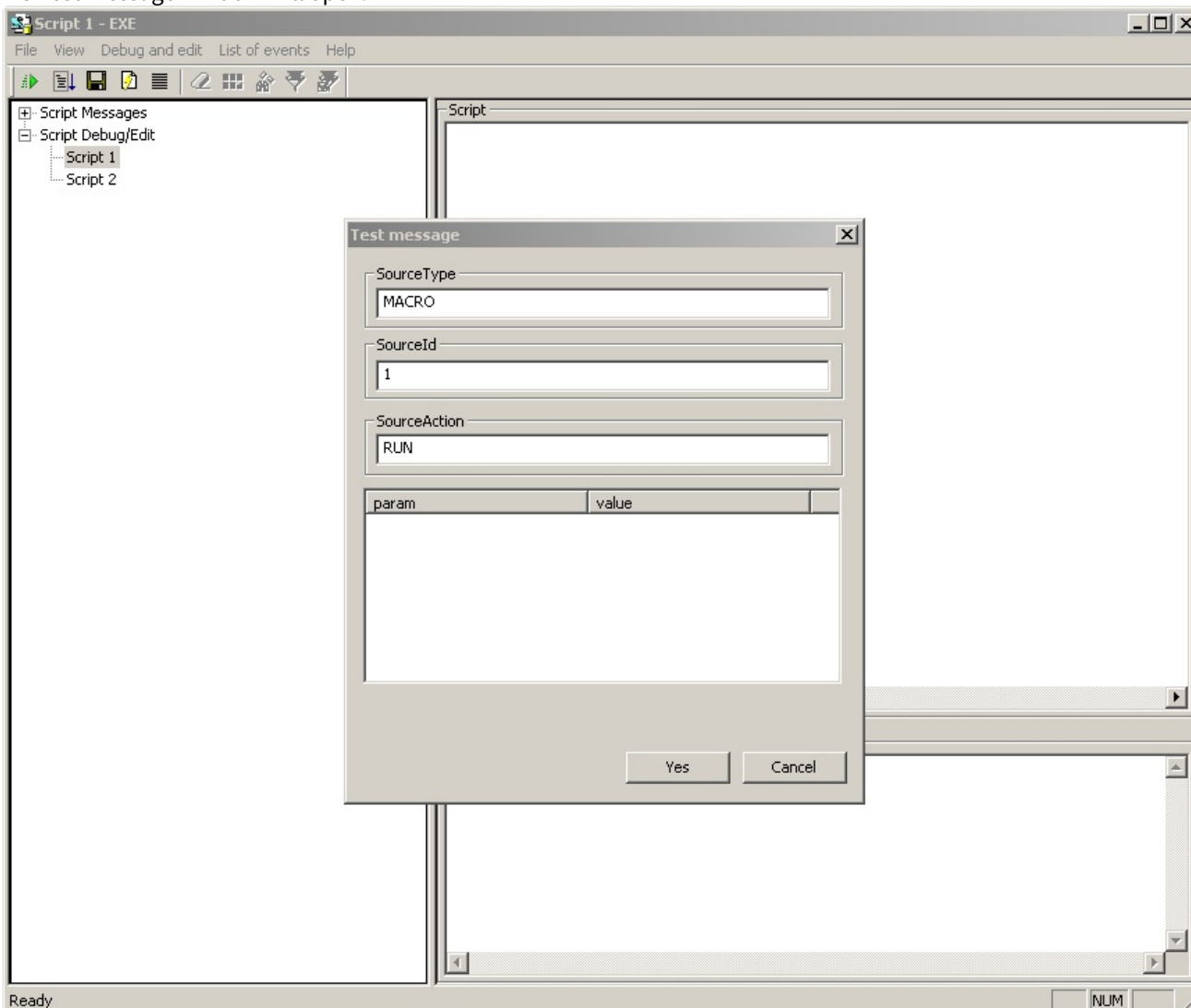
9.5.5.2.1 Creating test events

The *Editor-Debugger* utility is capable of generating test events chosen by the user to help debug the scripts. Test events are not registered by the video surveillance system, i. e. they are not listed in the events log and not saved to the database.

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

To create a test event, do the following:

1. In the **Debug and edit** menu, select **Edit test event**, or click the  button in the toolbar.
2. The Test message window will open.



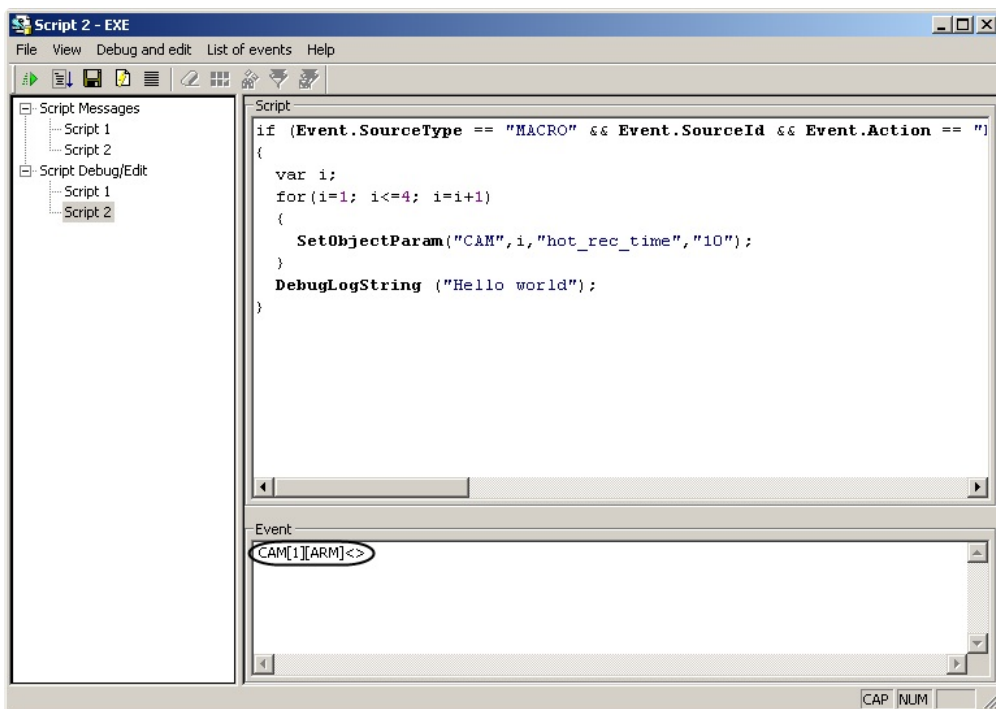
This window is used for entering the test event parameters.

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

The test event is now created.


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

Figure shows the test event example: “Arm Camera 111”.



9.5.5.2.2 Running the script with a test event

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

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

When the Test run in third-party debugger option is selected, the third-party debugger starts to run the test (see details in the **Using third-party debuggers** section).

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

9.5.5.3 Using debugger windows of the Editor-Debugger utility

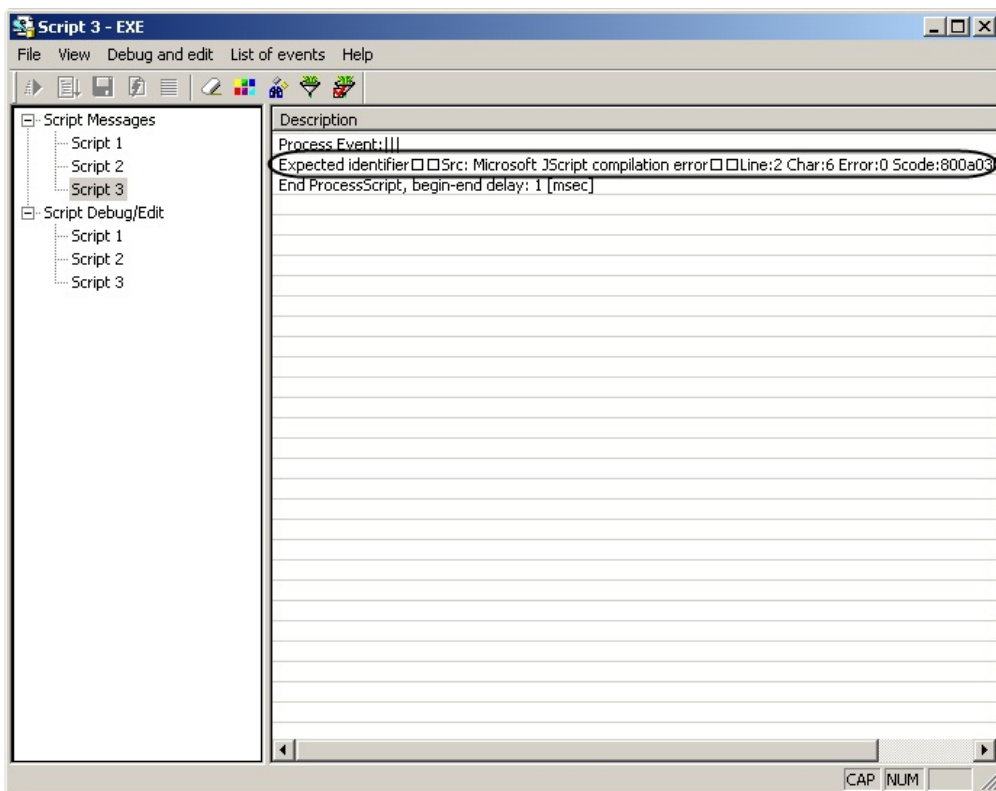
9.5.5.3.1 Displaying messages about starting, verifying, changing and executing scripts in the debugger windows

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

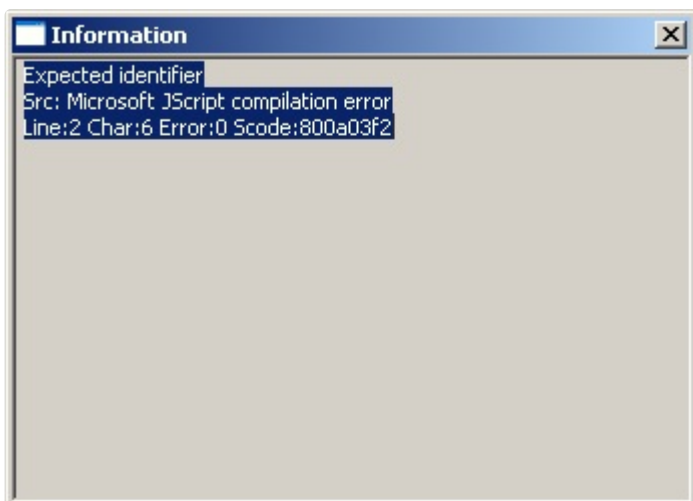
When the event occurs that triggers the script, the following message is displayed in the debugger window: “Process Event: <script triggering event>”; for example, if the script starts upon Macro 1 execution, the line reads “Process Event:MACRO|1|RUN|”.

At the moment of changing the script in *Debugger* utility or in the Intellect, the debugger window displays the message «Process Event: SCRIPT|script’s number|SETUP|» (for example, while changing the script with number 1, the debugger window displays «Process Event: SCRIPT|1|SETUP|»).

Script syntax is checked before execution. In case of syntax errors, related error messages will be displayed in the debugger window. Figure shows an example of a syntax error message.



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



The message contains the following information:

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

In case no errors were detected, the following message will be displayed in the debugger window: **Script first run OK**. Then, the script will run.

Script runtime errors are also displayed in the debugger window.

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

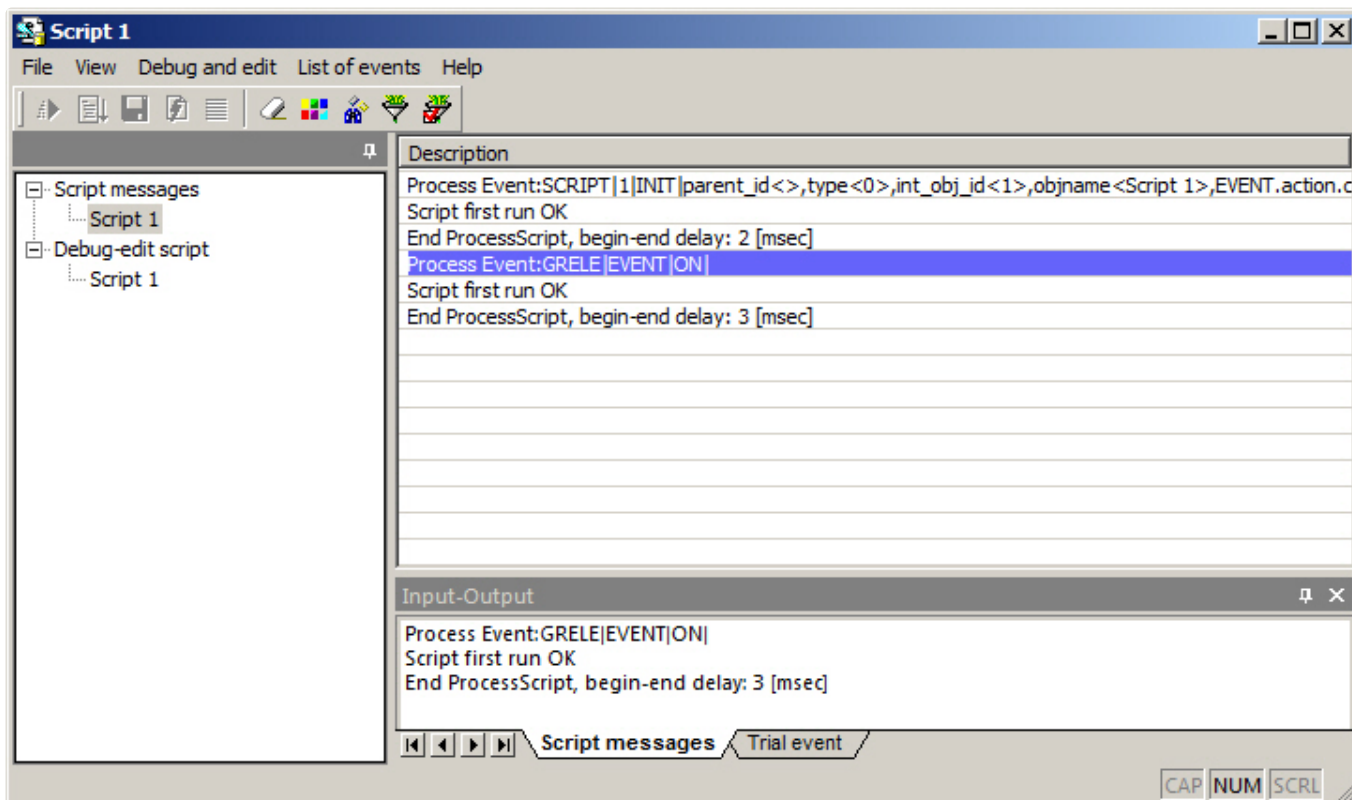
9.5.5.3.2 Viewing the script messages


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

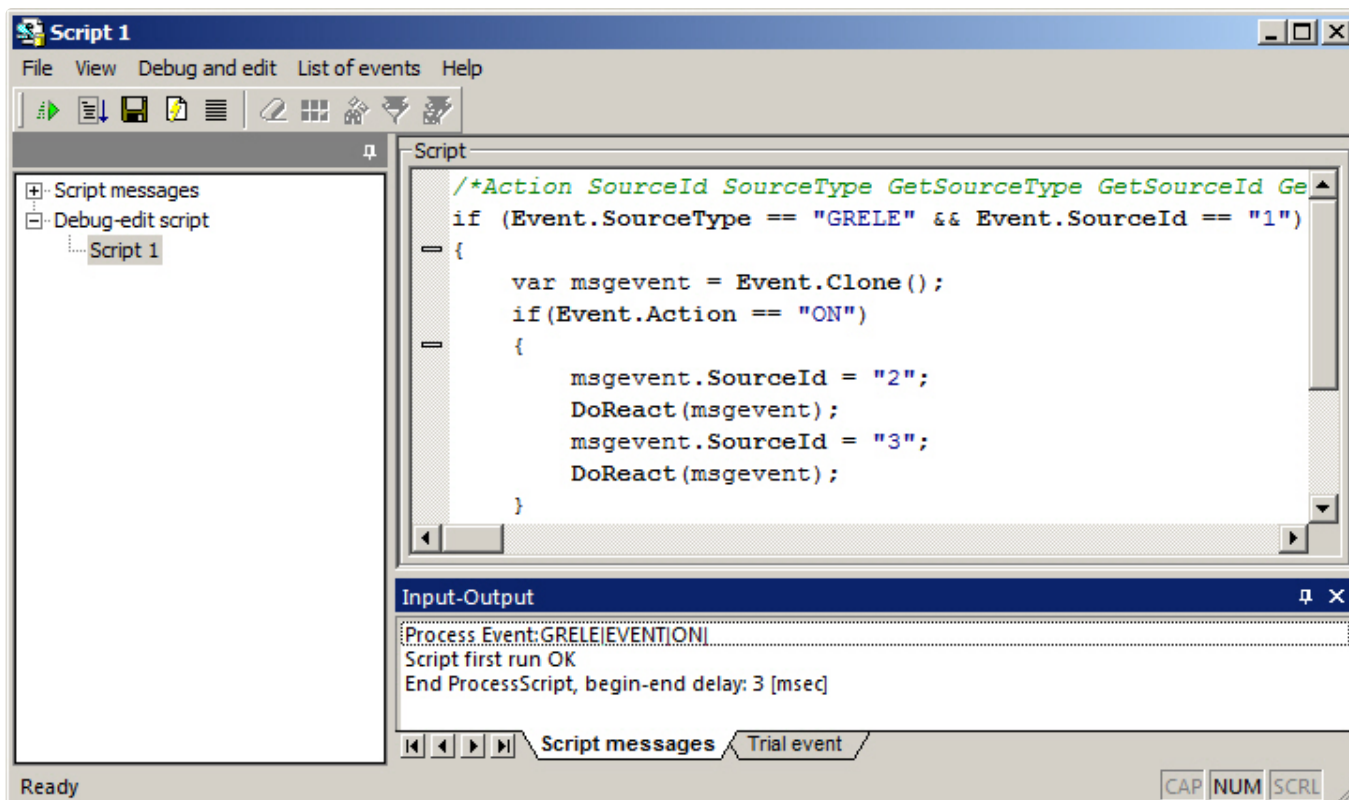
A separate debugger window is assigned to each script in the *Editor-Debugger* utility.

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

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



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



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

9.5.5.4 Using third-party debugger programs

The *Intellect* software package officially supports debugger *Microsoft Visual Studio 2005*.

The *Intellect* software package allows using third-party debuggers for processing JScript scripts. These programs may have the functionality that is not included in the *Editor-Debugger* utility, for example, step-by-step script execution, watching script variables during script execution, etc.

Note

We do not recommend using third-party debuggers, since they do not provide full compatibility with the *Intellect* software, and may lead to failure of the *Intellect* software

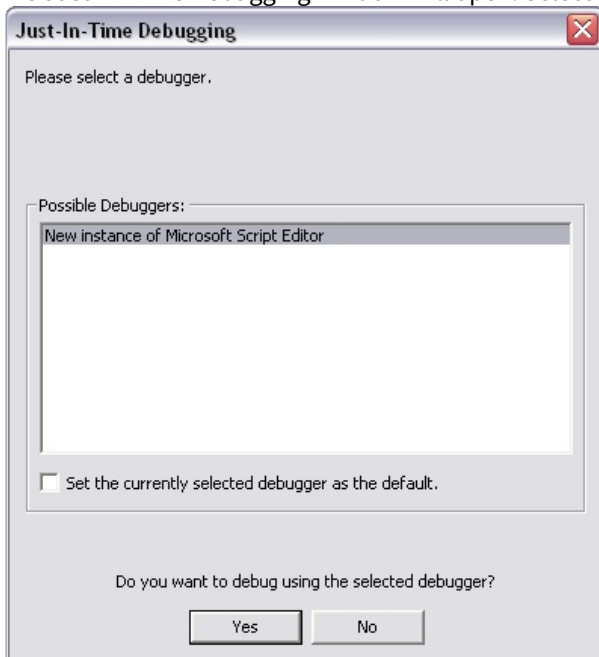
We strongly recommend introducing the breakpoint in the script when using third-party debuggers. To insert the breakpoint, add the following line to the script: "debugger;". The script execution will pause at this line, and the debugger will start.

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

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

1. Create the script and insert the "debugger;" line in it.
2. Create a test event for the script.
3. In the **Debug and edit menu**, select **Test run in third-party debugger**.

4. The Just-In-Time Debugging window will open. Select one of the debuggers installed on the computer.



5. Click **Yes** to confirm the selection.

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

Example. A script with the use of the breakpoint when Macro 1 starts.

```

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

```

9.5.6 Examples of scripts in JScript language

On the page:

- [Example 1. Visualisation of operating the Queue length detection in the Video Surveillance monitor](#)
- [Example 2. Visualisation of operating the People Counter detection in the Video Surveillance monitor](#)
- [Example 3. Setting the text for displaying on the map](#)
- [Example 4. Displaying camera on the monitor by](#)

clicking the button on the remote control

- Example 5. Sending command to camera using the camera HTTPAPI
- Example 6. Script for framing abandoned objects in live video
- Example 7. Using START and STOP events for the Failover Service
- Example 8. Using the embedded visitors counter on Bosch FLEXIDOME IP dynamic 7000 VR IP camera
- Example 9. Sending e-mail with HTML markup
- Example 10. Creating test users
- Example 11. Changing the status of the event in the Incident manager

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

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

The **Queue length detection** object (part of the Detector Pack package), **Camera** and **Captioner** objects (instead of N, M, L symbols set the corresponding numbers of the Queue length detection, Camera and Captioner objects) are to be created and configured for correct script working in the *Intellect* software.

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

As a result the text message about the current queue length will be imposed on the video image while displaying the corresponding camera in the Video Surveillance monitor.

Settings of the font, color and position of text is configured on the settings panel of the **Captioner** object.

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.

9.5.6.2 Example 2. Visualisation of operating the People Counter detection in the Video Surveillance monitor

The **People Counter Detection** object (part of the Detector Pack package), **Camera**, **Captioner** and **Macro** objects (instead of N, M, L, P symbols set the corresponding numbers of the People counter detection, Camera, Captioner and Macro objects) are to be created and configured for correct script working in *Intellect*.

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

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

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

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

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

As a result the text message about number of entering and exiting people will be imposed on the video image while displaying the corresponding camera in the Video Surveillance monitor.

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.

Settings of the font, color and position of text is configured on the settings panel of the **Captioner** object (see [Configuring captions display on a video image](#)).

Previously create the **Macro** object on the **Programming** tab to null the counter of visitors. The name of the object can be changed to “Null of people counter” for ease of using.

Macro of setting to zero can be run manually by main menu of *Intellect* or automatically in any specified time (use the **Events** table on the settings panel of the **Macro** object where the previously configured **Time zone** object is to be specified). Detailed information about using the **Macro** and **Time zone** objects is given in the [Administrator's Guide](#) document).

9.5.6.3 Example 3. Setting the text for displaying on the map

It is possible to select the type of **Text** displaying while attaching object to the map (see [Attaching objects to the layers of interactive map](#)). Also the script on the JScript language can be used for changing the displayed text.

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

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

9.5.6.4 Example 4. Displaying camera on the monitor by clicking the button on the remote control

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

Behavior	Action	Param.
Left button	(Default)	
Right button	(Default)	
Middle button	(Default)	
Scrolling	(Default)	
Shift + Scrolling	(Default)	
Shift + Left	(Default)	
Shift + Right	(Default)	
Shift + Middle butt.	(Default)	

Butt.	Action	Param.
1	Go to preset	1
2	Go to preset	2
3	Go to preset	3
4	Go to preset	4
5	Go to preset	5
6	Go to preset	6
7	Go to preset	7
8	Go to preset	8
9	Go to preset	9
10	Go to preset	0

Example. When the button is clicked on the control panel, display corresponding camera in the active monitor. The script is to timer trigger with ID=1.

Note

The **Timer** object is to be created and configured beforehand and the current year is to be set. Information on how to configure the Timer object can be found in [Creating and configuring the Timer object](#).

After each button click on the control panel wait for 2 seconds until clicking another button. If there is no button click, then the camera with dialed number is to be displayed.

```

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

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

```

9.5.6.5 Example 5. Sending command to camera using the camera HTTPAPI

Example. Camera IP address is 192.168.0.13.

The command enables the camera wiper:

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

The command disables the camera wiper:

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

These commands are to be sent to camera using the JScript script.

```
function DoPreset(preset)
{
    xmlhttp=new ActiveXObject("MSXML2.XMLHTTP");
    if(xmlhttp == null)
    {
        return;
    }
    xmlhttp.open("GET", "http://192.168.0.13/httpapi/SendPTZ?action=sendptz&PTZ_PRESETSET="+pre
set, false,"admin","1234");

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

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

9.5.6.6 Example 6. Script for framing abandoned objects in live video

If the object tracking function is in use (see [Creating and configuring the Timer object](#)), then detected objects will be framed in video when viewing the archive. To frame abandoned objects in live video use the script aimed to frame an abandoned object when receiving alarm from VMDA detection tool:

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

```

else
{
    DoReactStr("MONITOR", "", "DEL_MARKRECT", "cam<"+cam+">, id<"+cam+">");
}
}

```

9.5.6.7 Example 7. Using START and STOP events for the Failover Service

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

```

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

        if (!(id==pid)) {
            DoReactStr("FAILOVER",pid,action,"");
        }
    }
}
}

```

9.5.6.8 Example 8. Using the embedded visitors counter on Bosch FLEXIDOME IP dynamic 7000 VR IP camera

Upon reaching the number of visitors equal to 20 on the embedded visitors counter of the Bosch FLEXIDOME IP dynamic 7000 VR IP camera (with ID 1), call the macro 1.

```

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

```

9.5.6.9 Example 9. Sending e-mail with HTML markup

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

Face detected

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

```

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

var test_event = CreateMsg();
test_event.StringToMsg("MAIL_MESSAGE|1|SEND_RAW|cc<>,to<daniel@axxonsoft.com>,objname<Mail
message 1>,subject<>,parent_id<1>,flags<>,pack<>,name<Mail message
1>,from<example@gmail.com>,_marker<>");

test_event.SetParam("body","<html><body>\r\n<h3>Face found</
h3>\r\n<table><tr>\r\n<td>Detected</td><td>Found in DB</td>\r\n</tr><tr>\r\n<td><img
width='200' alt='image1' src='cid:"
+
file1
+
"/></td>\r\n<td><img width='200' alt='image2' src='cid:"
+
file2
+
"/></td>\r\n</tr><tr>\r\n<td>E-mail sent from Intellect</td>\r\n</tr></table>\r\n</body></
html>");
test_event.SetParam("attachments",file_folder+file1+";" + file_folder+file2);
test_event.SetParam("is_body_html", 1);
DoReact(test_event);
}

```

9.5.6.10 Example 10. Creating test users

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

Note

For more information on access levels and access cards, see the *ACFA Intellect* documentation in the [AxxonSoft documentation repository](#).

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

```

dep=10; // department ID
start=100; // first user ID
last=150; // last user ID
acc_lev=1; // access level ID

```

```

dep_count=30; // max number of users in the department

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

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

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

```

9.5.6.11 Example 11. Changing the status of the event in the Incident manager

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

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

```

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

```

9.6 Appendix 1. Description of the Editor-Debugger utility

9.6.1 The purpose of the Editor-Debugger utility

The *Editor-Debugger* utility is designed for creating, debugging and editing scripts in the Intellect software package.

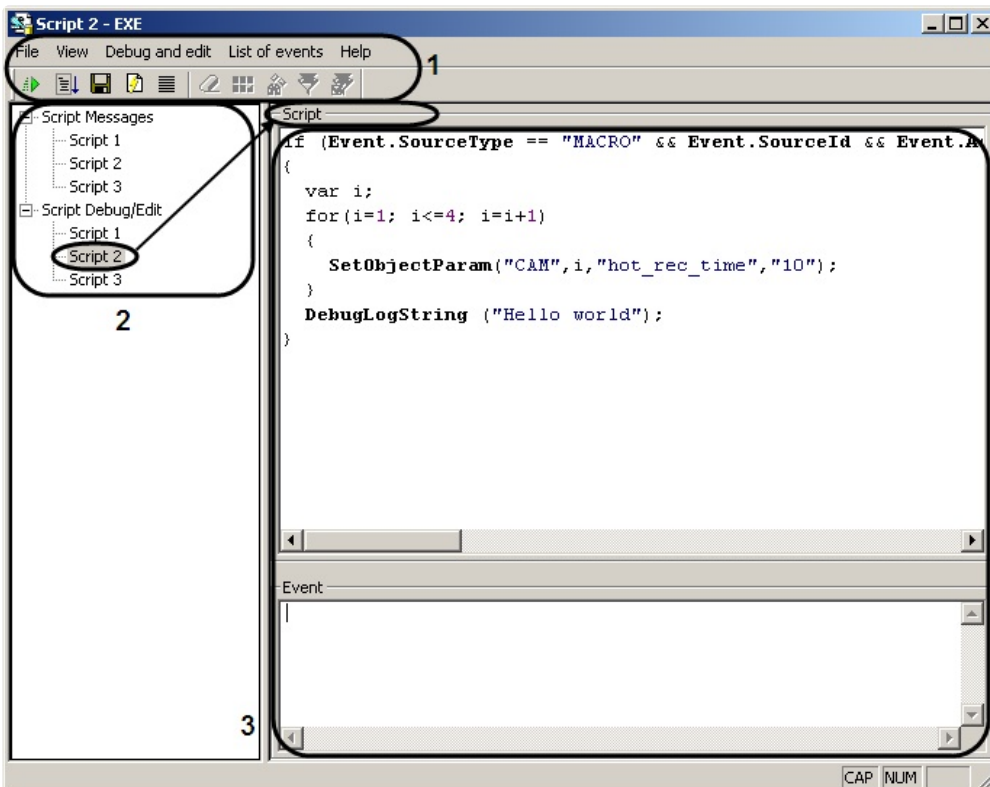
The *Editor-Debugger* utility provides the following functionality:

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

9.6.2 The interface of the Editor-Debugger utility

9.6.2.1 The Editor-Debugger window

The *Editor-Debugger* window contains the main menu, the toolbar (1), the object list (2) and the viewing/editing panel (3).



9.6.2.2 The Script Debug/Edit tab

9.6.2.2.1 Description of the Script Debug/Edit tab

The **Script Debug/Edit** tab is used for editing scripts and creating test events.

Figure shows the **Script Debug/Edit** panel.

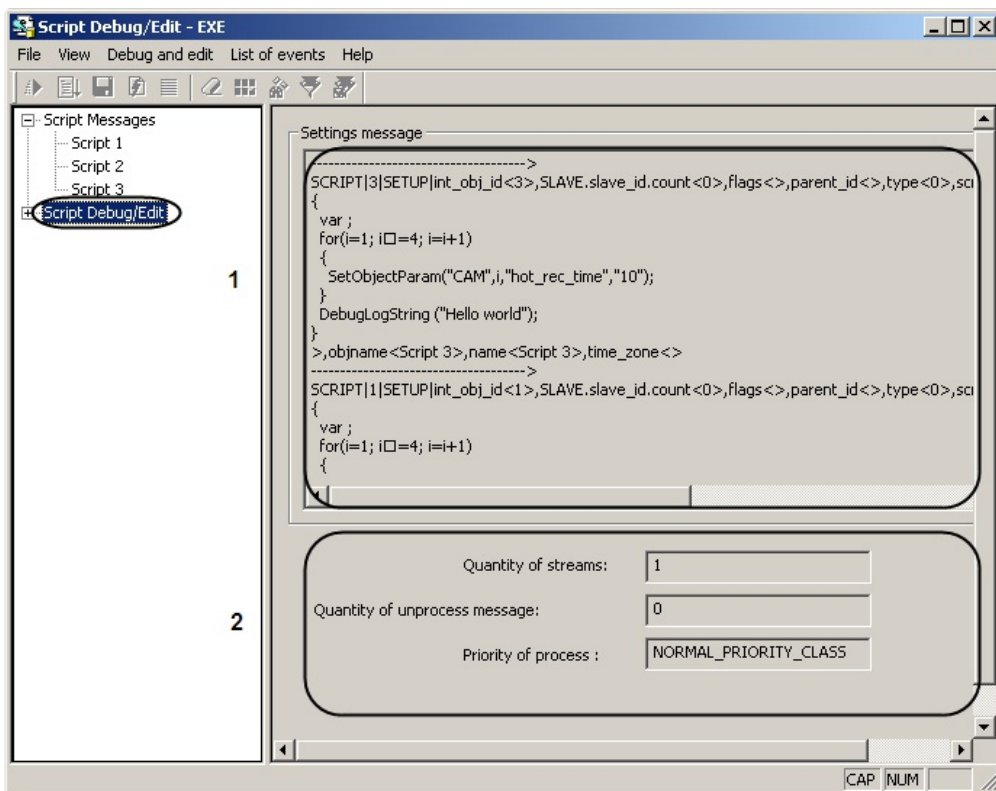


Table shows the description of the elements in the **Script Debug/Edit** panel.

№	Field name	Field type	Description	Units	Default value	Value range
1	Settings message	Auto	Script object initialization information	Latin, Cyrillic and special symbols	-	-
2	Additional information	Auto	Additional information about scripts	Latin, Cyrillic and special symbols	-	-

9.6.2.2.2 The Script object panel in the Script Debug/Edit tab

The **Script** objects in the **Script Debug/Edit** tab are used for creating and editing scripts and test events.

Figure shows the **Script** object panel.

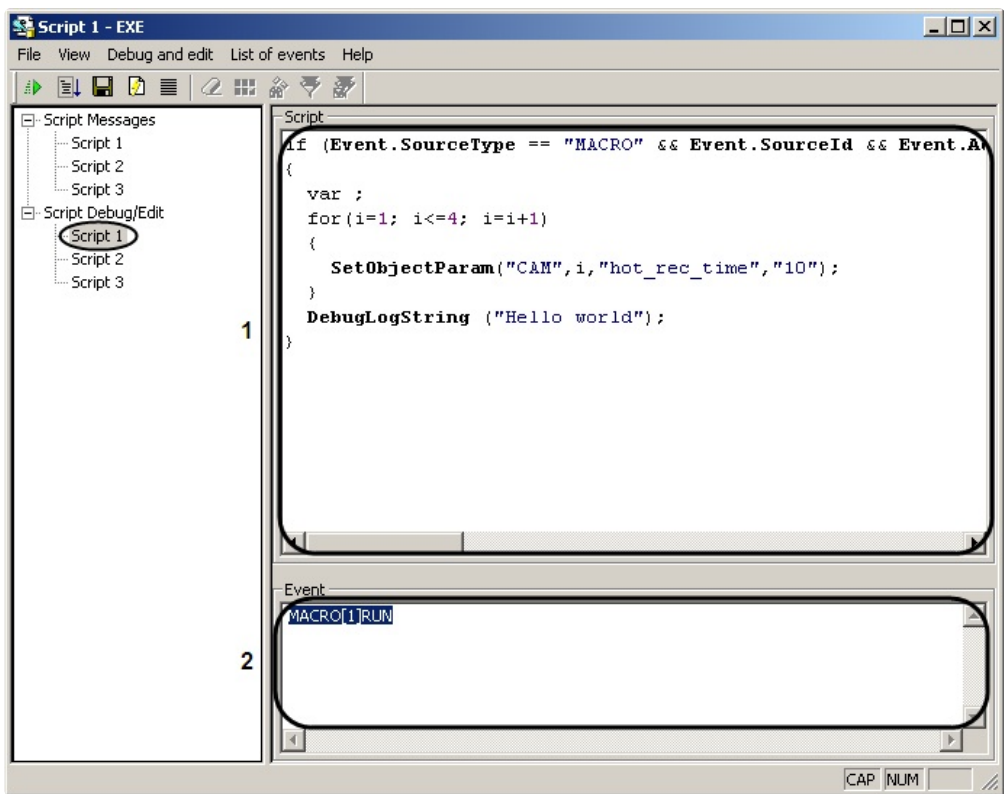


Table describes the elements in the **Script** object.

9.6.2.3 The Script Messages tab

9.6.2.3.1 Description of the Script Messages tab

The **Script Messages** tab is used to display the debugger windows for the scripts.

Figure shows the **Script Messages** panel.

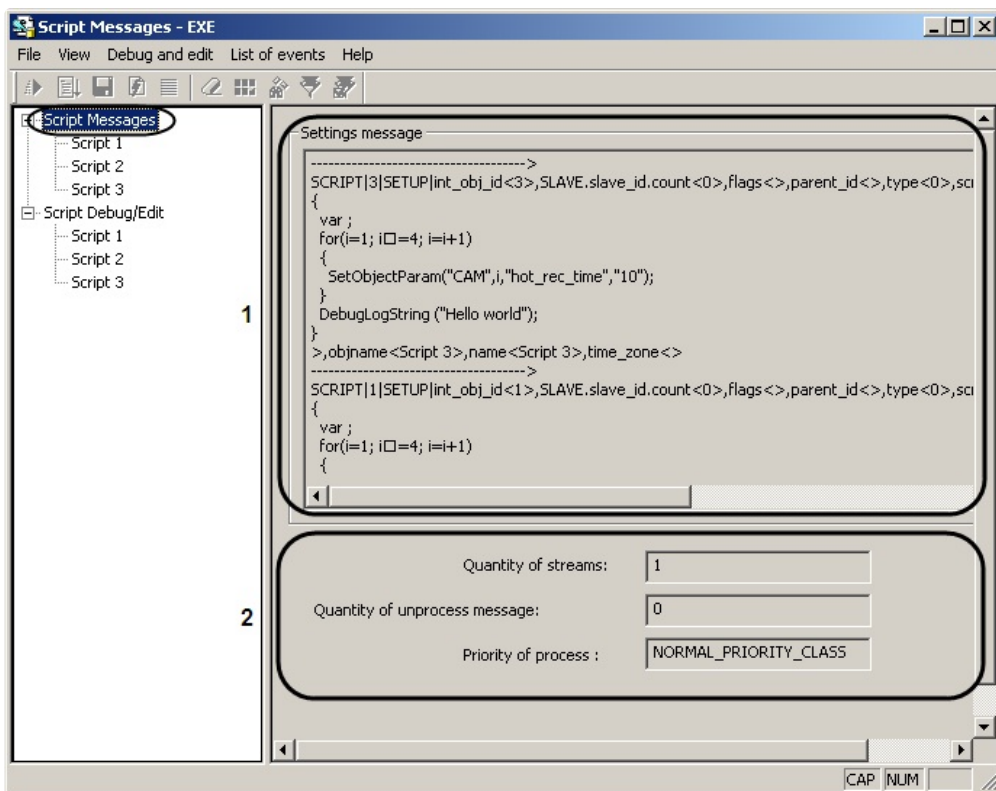


Table shows the description of the elements in the **Script Messages** panel.

№	Field name	Field type	Description	Units	Default value	Value range
1	Settings message	Auto	Script object initialization information	Latin, Cyrillic and special symbols	-	-
2	Additional information	Auto	Additional information about scripts	Latin, Cyrillic and special symbols	-	-

9.6.2.3.2 The Script object panel in the Script Messages tab

The **Script** panels in the **Script Messages** tab are used to display system, test and user events related to the scripts created in *Intellect*.

Figure shows the **Script** object panel.

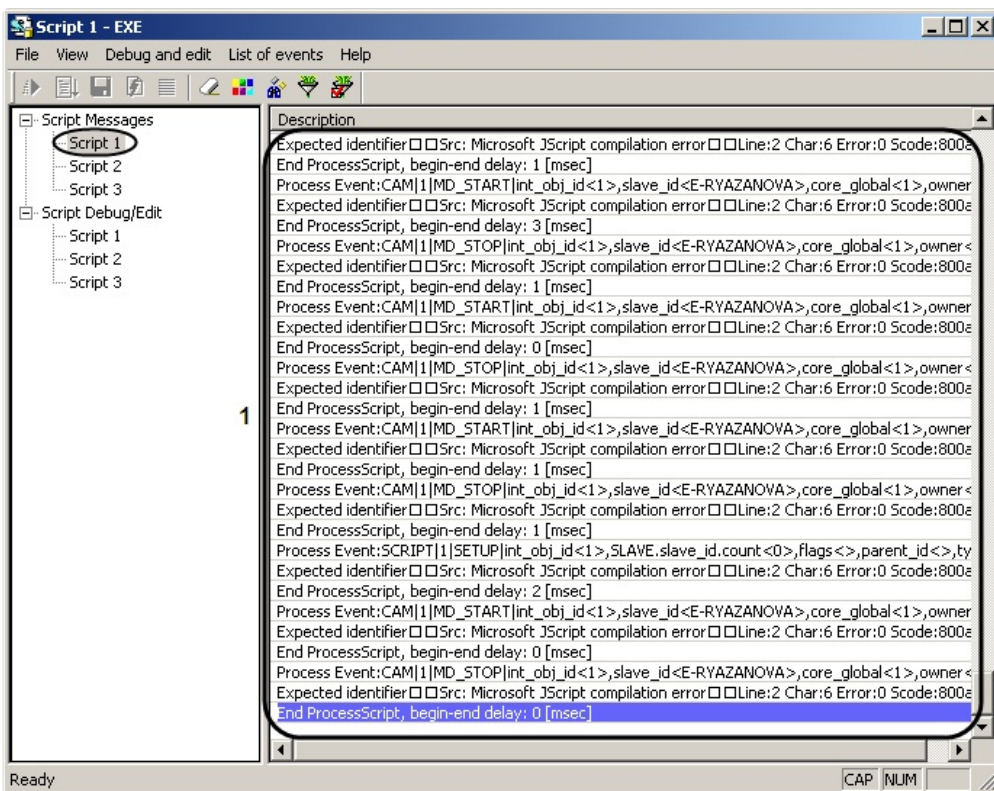


Table describes the elements in the **Script** object.

9.6.2.4 Main menu

On the page:

- [Description of the main menu](#)
- [The elements in the File menu](#)
- [The elements in the View menu](#)
- [The elements of the Debug and edit menu](#)
- [The Message list menu elements](#)

9.6.2.4.1 Description of the main menu

The *Editor-Debugger* main menu is used to call editing, debugging and other commands. The commands are grouped into functional menus: **File, View, Debug and edit, Message list** and **Help**.

Table describes the elements of the main menu.

No	Element name	Element type	Description	Units	Default value	Value range
1	File	Drop-down list of items	Commands for opening and saving scripts and closing the utility	-	-	
2	View	Drop-down list of items	Commands for displaying the toolbar and the status bar in the utility window	-	-	-
3	Debug and Edit	Drop-down list of items	Commands for script debugging	-	-	-
4	Message list	Drop-down list of items	Commands for changing the message display parameters in the debugger windows	-	-	-
5	Help	Drop-down list of items	The About command showing general information about the Editor-Debugger utility.	-	-	-

9.6.2.4.2 The elements in the File menu

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

Table describes the elements of the **File** menu.

No.	Element name	Element type	Description	Units	Default value	Value range
1	Save to database	Item	Saves the script in the object	-	-	-
2	Save to disk	Item	Saves the script into a text file on the hard drive			
3	Open from disk	Item	Opens the script file	-	-	-
4	Exit	Item	Shuts down the utility and closes the window	-	-	-

9.6.2.4.3 The elements in the View menu

The **View** menu contains commands for showing and hiding the toolbar and the status bar.

Table describes the elements of the **View** menu.

No	Element name	Element type	Description	Units	Default value	Value range
1	Toolbar	Checkbox	Shows or hides the toolbar	Boolean	Checked	Check – show toolbar Uncheck – hide toolbar
2	Status bar	Checkbox	Shows or hides the status bar	Boolean	Checked	Check – show status bar Uncheck – hide status bar

9.6.2.4.4 The elements of the Debug and edit menu

The **Debug and edit** menu contains the commands for debugging scripts.

Table describes the elements of the **Debug and edit** menu.

No	Element name	Element type	Description	Units	Default value	Value range
1	Test run	Item	Runs the script with the test event	-	-	-
2	Test run in third-party debugger	Item	Runs the script using the third-party debugger	-	-	-
3	Edit test event	Item	Opens the window for editing test events	-	-	-
4	Summary information	Item	Opens the Thread Information window showing system, test and user messages related to the current script	-	-	-
5	Go to line	Item	Opens the window for entering the script line and character number to go to	-	-	-

9.6.2.4.5 The Message list menu elements

The **Message list** menu contains commands for changing the message display parameters in the debugger window.

Table describes the elements of the **Message list** menu.

9.6.2.5 The Filter dialog window

The Filter window allows to set the filtering criteria for the messages displayed in the **Description** field of the debugger window.

To open the Filter window, do one of the following:

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

Figure shows the Filter window.

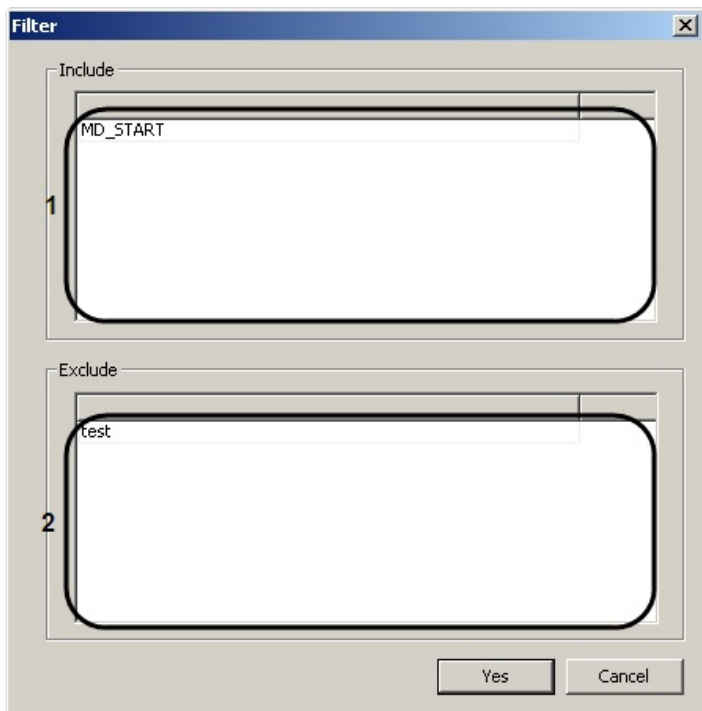


Table describes the elements in the Filter window.

9.6.2.6 The Color dialog window

The **Color** dialog window is used for setting up color highlighting of the lines containing certain words.

To open the **Color** window, do one of the following:


1. Click the Colors () button in the *Editor-Debugger* toolbar.
2. In the **Message list** menu, select **Colors**.

Figure shows the Color window.

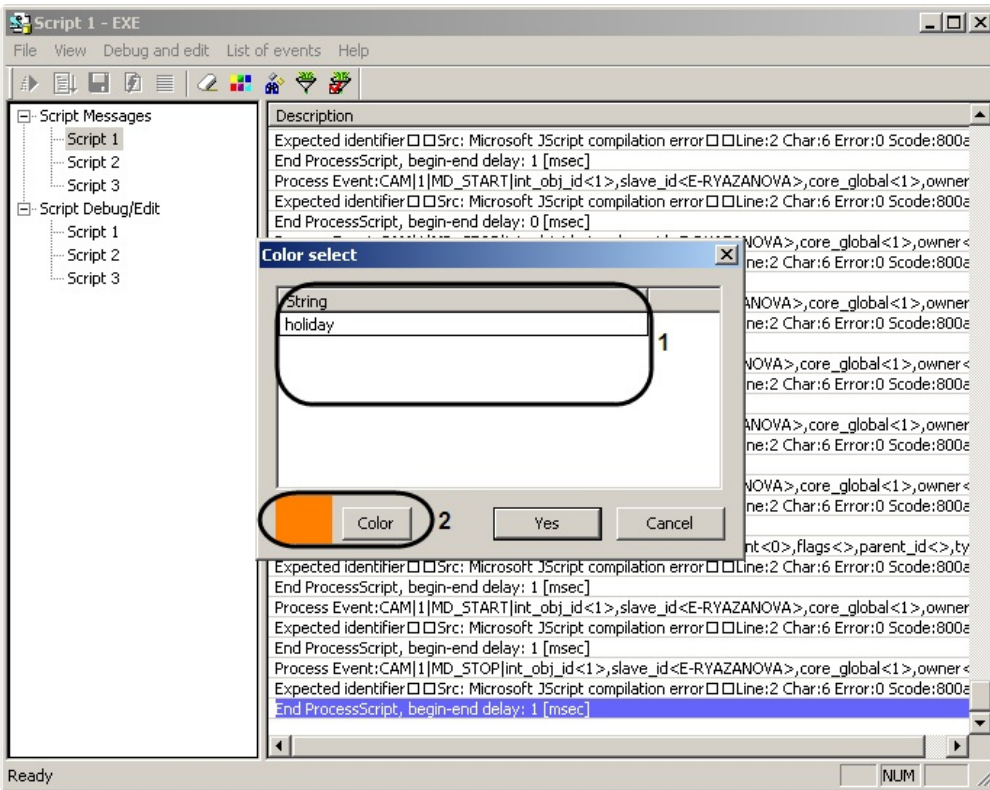


Table describes the elements in the **Color** window.

9.6.2.7 The toolbar of the Editor-Debugger utility

The *Editor-Debugger* toolbar is used for calling frequently used functions of the utility.

The toolbar operates in two modes: when the script control buttons are active, or when the debugger window control buttons are active.

The mode depends on the currently active tab of the *Editor-Debugger* utility: either the **Script Debug/Edit** tab for editing scripts, or the **Script Messages** tab for viewing messages in the debugger window.

Figure shows the toolbar in the script editing mode.

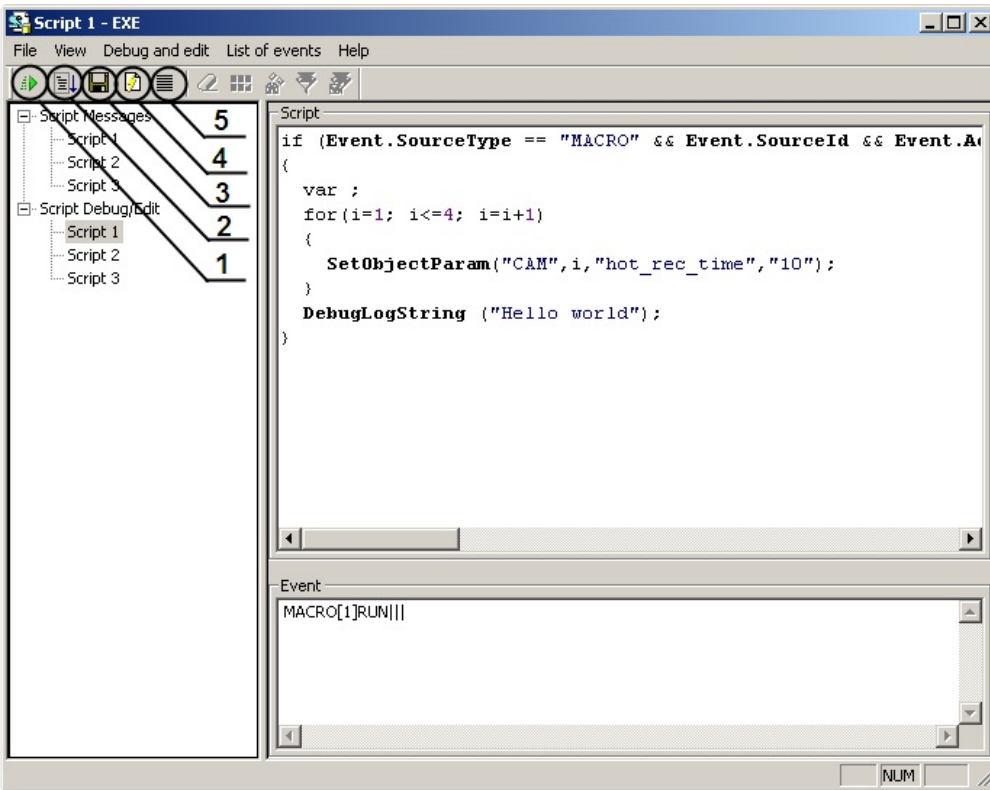


Table shows the elements of the toolbar in the script editing mode.

Figure shows the toolbar in the script debugging mode.

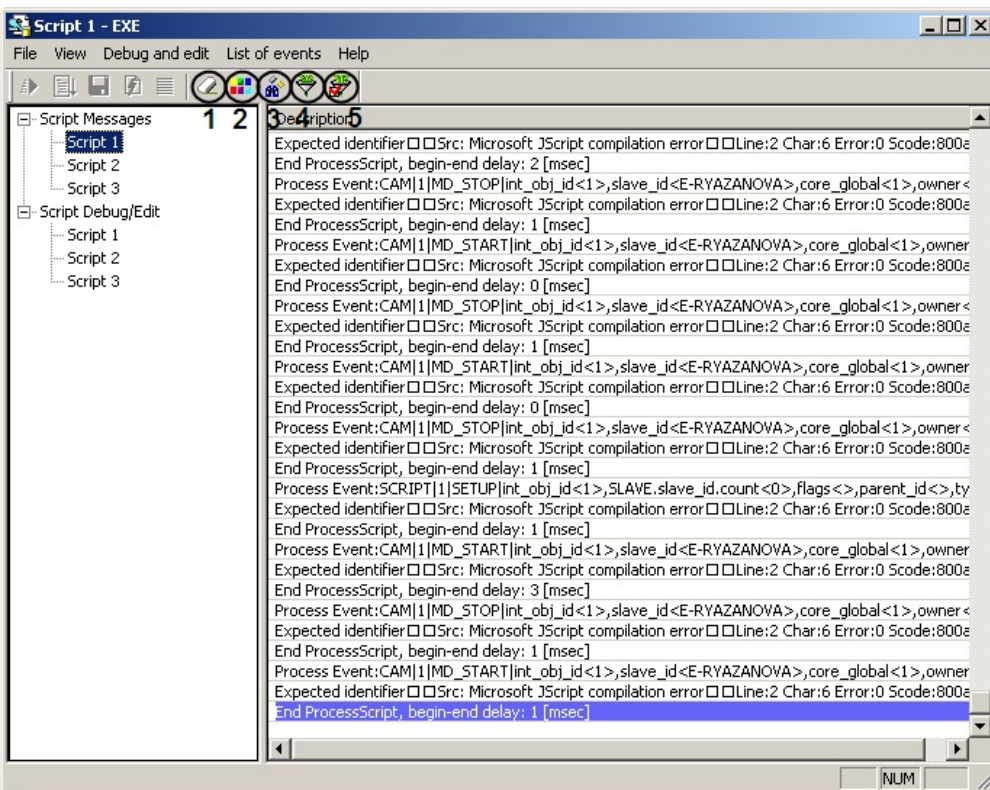


Table shows elements of the toolbar in the script debugging mode.

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

9.7.1 Purpose of virtual objects and their implementation in Intellect

Virtual objects represent software emulation of new *Intellect* objects and allow configuring their states, reactions and events. Virtual objects are handled using scripts, macros and macroevents.

Virtual objects are created using the `ddi.exe` and `CustomTypeEditor.exe` utilities in the `<Intellect installation folder>\Tools`.

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

The procedure for creating and configuring a virtual object:

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

9.7.2 How to create a virtual object

Here one can find out how to create the following virtual objects:

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

An object of the CUSTOM type has the property set:

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

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

9.7.2.1 DBI file preparation

A DBI file is prepared using the `ddi.exe` utility. Details on how to handle it can be found in [The ddi.exe utility for editing database templates and external settings files](#) section of [Administrator's Guide](#).

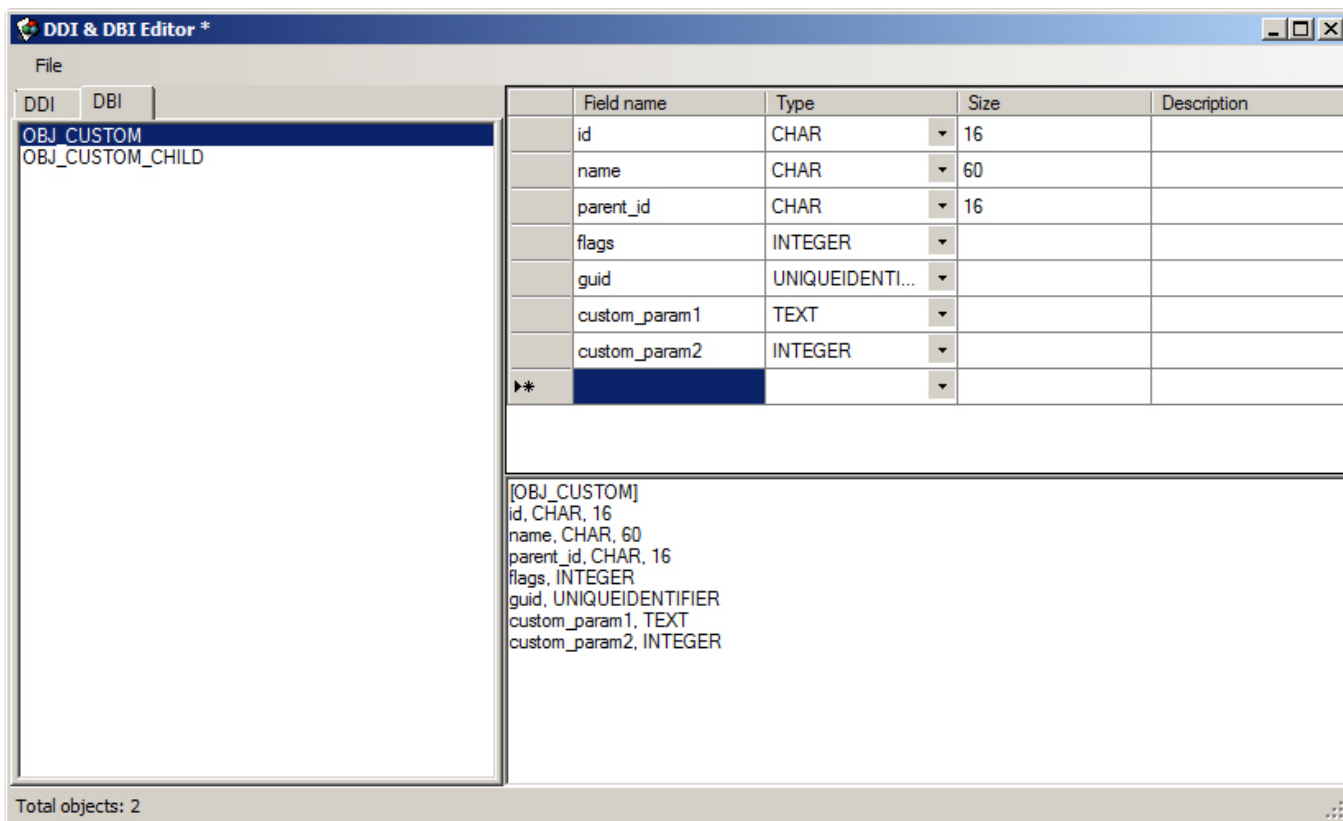
A DBI file for the objects of the CUSTOM and CUSTOM_CHILD type is created as follows:

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



Attention!

Object (table) names should look like OBJ_<object type>.



- Set the parameters for each object. The **id**, **name**, **parent_id**, **flags**, **guid** parameters are mandatory for all objects. **Custom_param1**, **custom_param2** in the example in the figure are custom parameters. You can also set other parameters used in *Intellect*. For example, adding the **region_id** parameter will allow you to set areas and regions for an object (see [Subdivision of the protected facility into areas and regions](#)).
- Save the changes using the **Save** command in the **File** menu. The saved file must have the dbi extension and must be located in *Intellect* installation directory, for example, C:\Program Files (x86)\Intellect\intellect.custom.dbi

DBI file preparation is complete.

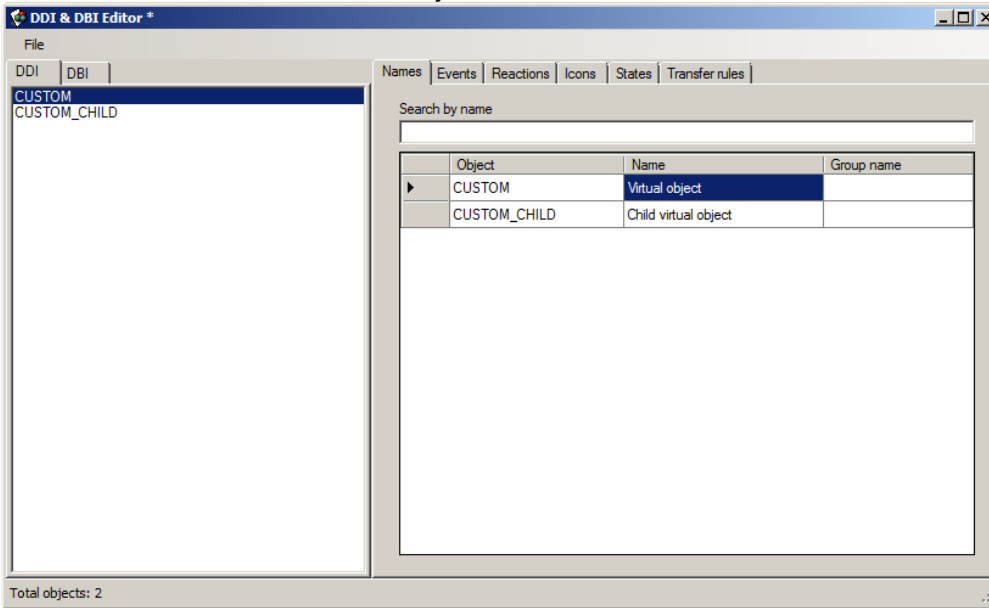
9.7.2.2 DDI file preparation

A DDI file is prepared using the ddi.exe utility. Details on how to handle it can be found in [The ddi.exe utility for editing database templates and external settings files](#) section of [Administrator's Guide](#).

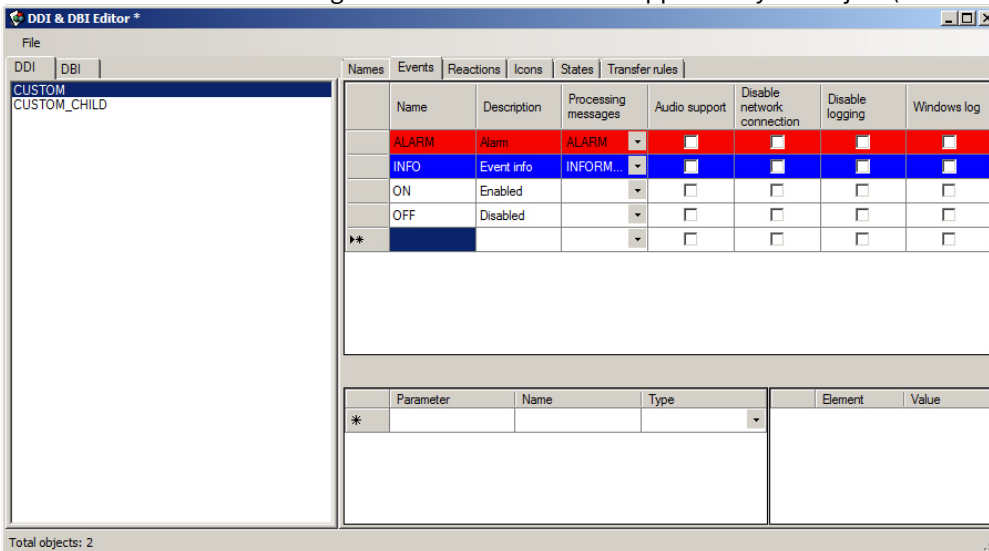
A DDI file for CUSTOM and CUSTOM_CHILD object types is created as follows:

- Run ddi.exe (see [The ddi.exe utility for editing database templates and external settings files](#)).

2. Create CUSTOM and CUSTOM_CHILD objects in the **DDI** tab as shown below.



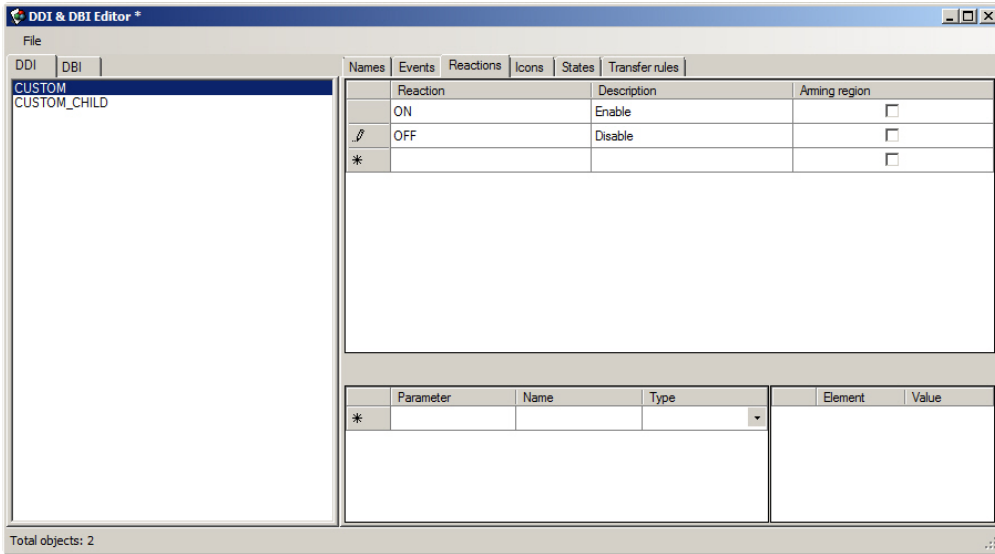
3. Go to the **Events** tab and configure events that are to be supported by the object (see the figure).



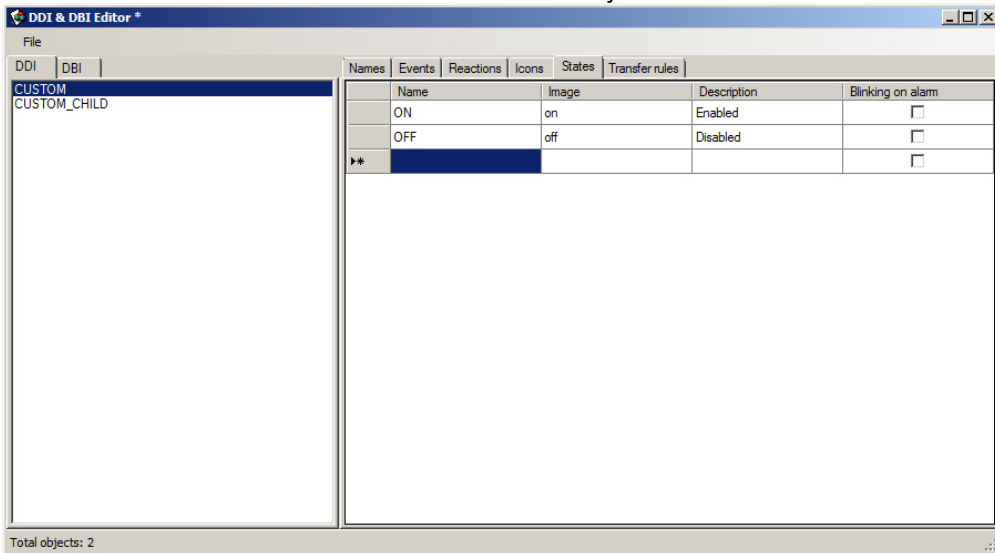
4. Go to the **Reactions** tab and configure reactions that are to be supported by the object (see the figure).

Note

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



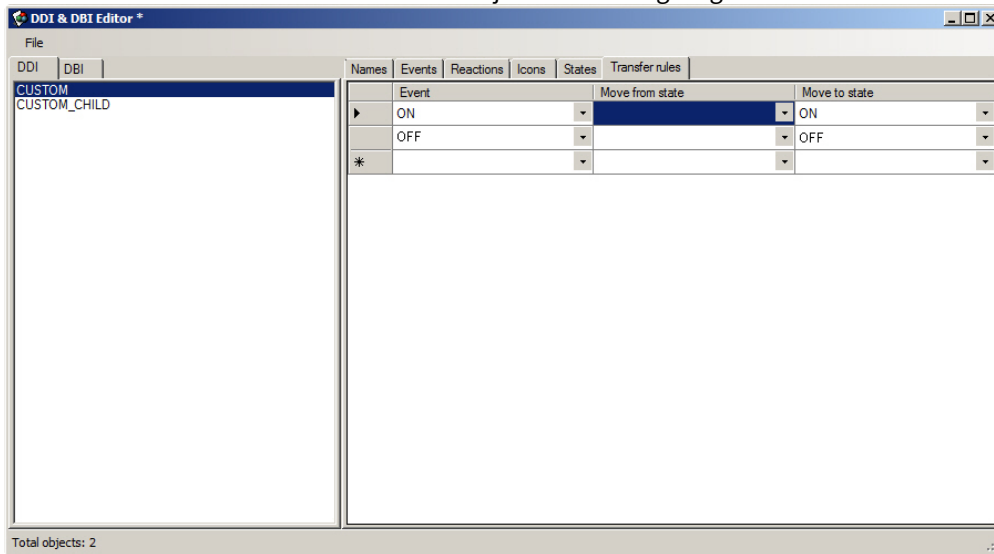
5. Go to the **States** tab and describe the states that the object can take. Here there are two states – ON and OFF.



Note.

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

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



Transition rules is a simple state machine – an event is an input action and a state is a result.

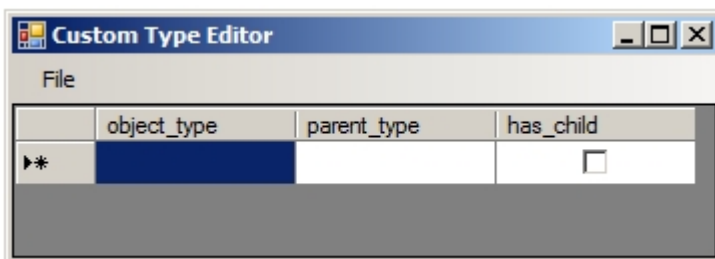
An unconditional transition is used in this case: if CUSTOM|ON event is received, then there is transition to the ON state, if CUSTOM|OFF event is received, then there is transition to the OFF state.

7. To save changes use the **Save** command in the **File** menu. The saved file must have the ddi extension and be stored in the folder corresponding to the required language, e.g. C:\Program Files (x86)\Intellect\Languages\en\intellect.custom.ddi

The DDI file preparation is completed.

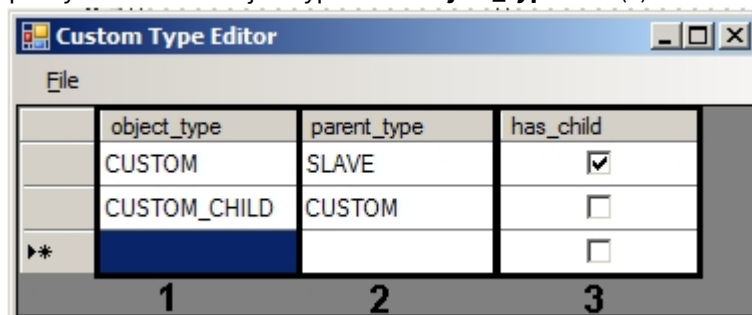
9.7.2.3 XML file preparation

An XML file is prepared using the CustomTypeEditor.exe utility that can be found in <Intellect installation directory>\Tools. The utility overview is shown below.



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

1. Specify the name of object type in the **object_type** field (1).



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

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

```
<?xml version="1.0" standalone="yes"?>
<objects>
<object>
<object_type>CUSTOM</object_type>
<parent_type>SLAVE</parent_type>
<has_child>1</has_child>
</object>
<object>
<object_type>CUSTOM_CHILD</object_type>
<parent_type>CUSTOM</parent_type>
</object>
</objects>
```

Modify it manually if required.

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

9.7.2.4 Creating and using a virtual object in Intellect

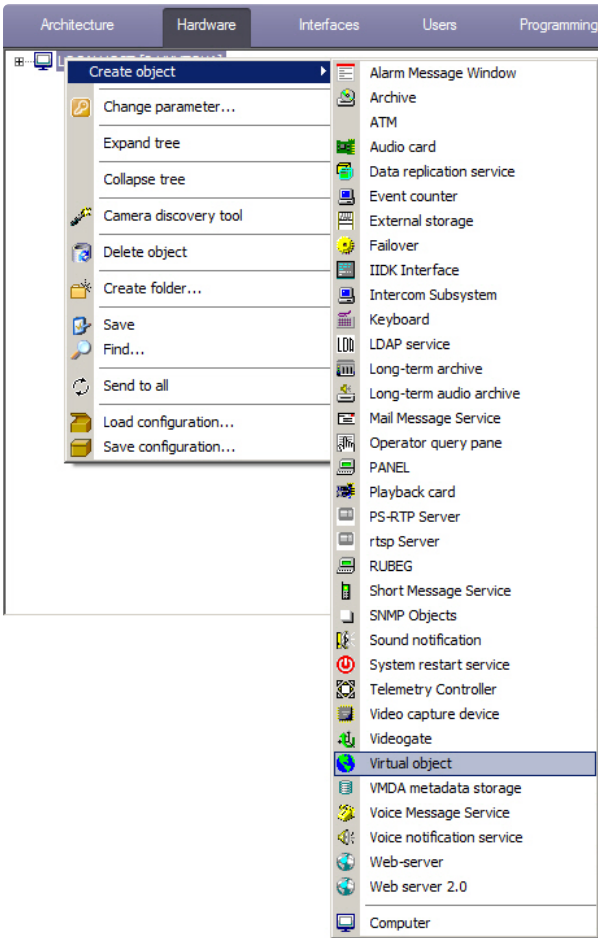
On the page:

- [Displaying on the map](#)
- [Using in macros](#)
- [Sample program in JScript to change the state of a virtual object](#)

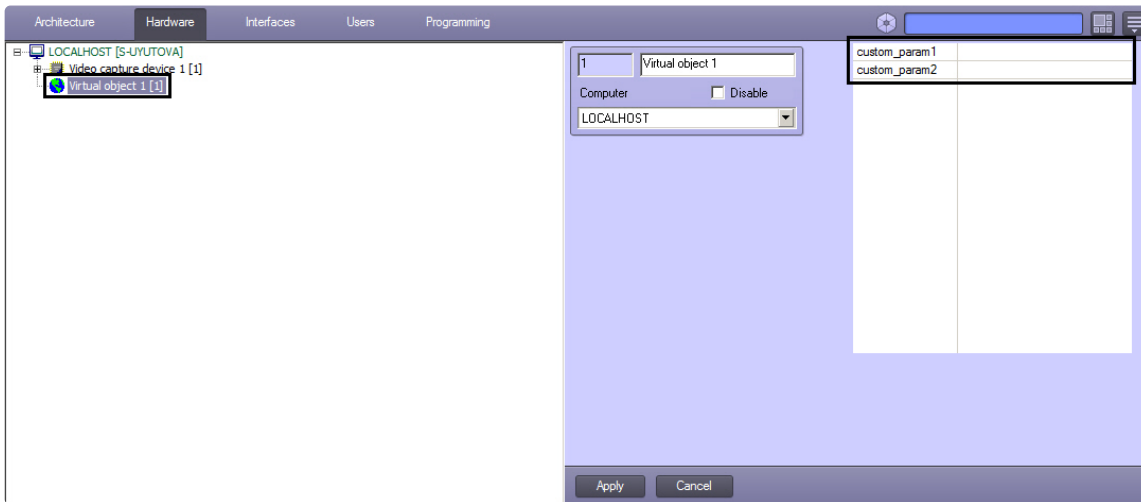
Attention!

After preparing the required files and before creating virtual objects in *Intellect*, it is necessary to update the main database using [The idb.exe utility for converting databases, selecting database templates and making backup copies of databases](#).

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

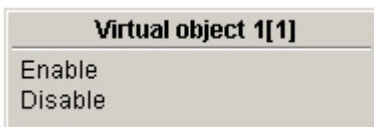


Custom parameters are displayed on the settings panel of the created virtual object—custom_param1 and custom_param2 in this example. Their values can be set in the table.



9.7.2.4.1 Displaying on the map

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



9.7.2.4.2 Using in macros

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

2 Macro 2 Disable

Settings

State Local Hidden

Type	Nu...	Name	Event
Macro	1	Macro 1	Action executed

Name	Value
------	-------

Type	Nu...	Name	Action
Virtual object	1	Virtual object 1	Enable

Name	Value
------	-------

Apply Cancel

Note

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

9.7.2.4.3 Sample program in JScript to change the state of a virtual object

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

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

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
    var msgevent = CreateMsg();
    msgevent.SourceType = "CUSTOM";
    msgevent.SourceId = "1";
    msgevent.Action = "ON";
    NotifyEvent(msgevent);
}
```

9.8 Programming Guide (JScript). Conclusion

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

1. [Installing and configuring security system components guide.](#)
2. [Operator's Guide.](#)
3. [Administrator's Guide.](#)

If while operating the given software product you have faced difficulties and problems, 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 occurred?
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 how to improve our software and documentation.

10 Description of events and reactions of system objects

All reactions for the main objects of the 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 [Getting the list of object names, reactions and events in INTELLECT™](#)).
2. Viewing events for the selected object of system using the control panel of the **Macro** system object (see [Creating and configuring Macro events](#)).

10.1 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.

Event	Description
"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
"SETUP" – sets parameters of video capture device.

Command – command description
"SET_DRIVES" – sets 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. <i>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 A, section SET_IPINT_PARAM reaction</i>
"START" – start playing video file in a virtual video capture device .
"STOP" – stop playing video file in a virtual video capture device .
"ENABLE" – enable object (uncheck the Disable box in the object settings panel)
"DISABLE" – disable object (check the Disable box in the object settings panel)

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).

- 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:\
}

```

- 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).

- 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
}

```

10.2 CAM

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

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

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

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

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

Events	Description	Comment
ARM	Camera is armed	If arming was performed by the Operator from the Map or Video surveillance monitor , the user_id<> parameter contains the identifier of the user who performed this action
ATTACH	Connecting	
BLINDING	Camera is sealed	
DETACH	Break	Event is generated when the input signal from the camera on the video capture device is lost
DISARM	Camera is disarmed	If disarming was performed by the Operator from the Map or Video surveillance monitor , the user_id<> parameter contains the identifier of the user who performed this action
FILE_REC_ERROR	Error of recording on disk	Event is generated when an error occurs when writing the video archive to disk
MD_START	Alarm	
MD_STOP	End of alarm	
PRINT	Print frame	
REC	Recording on disk	If recording was initiated by the Operator from the Map or Video surveillance monitor , the user_id<> parameter contains the identifier of the user who performed this action
REC_STOP	Stop recording on disk	If recording was stopped by the Operator from the Map or Video surveillance monitor , the user_id<> parameter contains the identifier of the user who performed this action
UNBLINDING	Camera is opened	
RECORDER_ON	Record is enabled	
RECORDER_OFF	Record is disabled	
DISC_MOUNT	Disk is mounted	
DISC_UNMOUNT	Disk is unmounted	

Events	Description	Comment
FINISHED_AVI_EXPORT	Video export is completed	<p>If the video export fails, the event has the non-null <code>error_result</code> parameter.</p> <p>In the <code>param<0></code> parameter there is additional information displayed in the corresponding column of the Event Viewer, in the following format: "ComputerName;ExportPeriod;UserName;UserID", for example, <code>param0<LOCALHOST;04-06-18 16:50:55_04-06-18 16:55:55;Smith;1></code></p>
MD_LIMIT	Object count in a frame exceeded	<p>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 <i>Administrator's Guide</i> 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.</p> <p>The <code>object_count<></code> parameter is the number of objects in the frame that the tracker detected, exceeds the specified limit, and differs from the previous value.</p> <p>See also the description of the <code>NEW_OBJECT</code> event below</p>
NEW_OBJECT	New object on a frame detected by tracker	<p>Among others, it contains the following parameters:</p> <ul style="list-style-type: none"> <code>total<></code> is the total number of objects in the frame at the time the event occurred <code>new_id<></code> is the identifier of the detected object
ARCH_DELETE	Record deletion	Deletion of the record from the camera archive from the Video surveillance monitor
OPEN_FILE	File opening	<p>Opening a file for playback by a virtual video capture device. Indicates that playback of the next file from the selected folder starts.</p> <p>Among others, the event contains the following parameters:</p> <ul style="list-style-type: none"> <code>name<></code> is the name of the file being played back <code>tss<></code> is the time in UTC format in milliseconds from 1/1/1970
CLOSE_FILE	File closing	<p>Finishing a file playback in a virtual video capture device. Indicates that playback of a file is ended.</p> <p>Among others, the event contains the following parameters:</p> <ul style="list-style-type: none"> <code>name<></code> is the name of the file that finished being played back <code>tss<></code> is the time in UTC format in milliseconds from 1/1/1970
ARCH_PROTECTED	File protected from rewriting	<p>The event is displayed when a user protects the file from being overwritten.</p> <p>The <code>param<0></code> parameter contains additional information about the computer name, fragment, user name and user id, which is displayed in the Add. Info column in the Event Viewer in the following format: "ComputerName;ProtectionPeriod;UserName;UserID", for example, <code>param0<LOCALHOST;04-06-18 16:50:55.612_04-06-18 16:55:55.612;Smith;1></code></p>
ARCH_UNPROTECTED	Rewrite protection removed from file	<p>The event is displayed when a user removes protection of the file from being overwritten.</p> <p>The <code>param<0></code> parameter contains additional information about the computer name, fragment, user name and user id, which is displayed in the Add. Info column in the Event Viewer in the following format: "ComputerName;ProtectionPeriod;UserName;UserID", for example, <code>param0<LOCALHOST;04-06-18 16:50:55.612_04-06-18 16:55:55.612;Smith;1></code></p>
FRAME_SKIPPED	Frames skipped	<p>The <code>FRAME_SKIPPED</code> event occurs in <i>Intellect</i> if there are frames skipped when recording to the archive. By default, the event is generated if there were more than 50 frame skips during the testing period. When frame skipping stops, the <code>FRAME_SKIPPED_STOP</code> event is generated. The description of this event contains information about the number of skipped frames and the time period in which there were skips. By default, both events are generated no more than once every 30 seconds.</p> <p>The events are controlled by the following registry keys (see Registry keys reference guide):</p> <ul style="list-style-type: none"> the <code>FRAME_SKIPPED</code> event can be disabled using the <code>FileSystem.NotifyCoreFrameSkipped</code> registry key; the delay time between the state changes in seconds is specified in the <code>FileSystem.RecordingStateChangeDelay</code> key;

Events	Description	Comment
FRAME_SKIPPED_STOP	End of frames skipping	<ul style="list-style-type: none"> the number of skipped frames for the period, during which the FRAME_SKIPPED event will be generated, is specified in the FileSystem.MaxSkippedFramesByPeriod key
ARCH_BOOKMARKED	Bookmark created	<p>The event is displayed when a user adds a bookmark.</p> <p>The param<0> (i.e. the Add. Info column in the Event Viewer) parameter contains the comment to the bookmark</p>
ARCH_UNBOOKMARKED	Bookmark deleted	The event is displayed when a user deletes a bookmark
TEMPERATURE_ALARM	Temperature threshold	The param0<> parameter contains the temperature value received from the thermal camera
IGNORE_KEEP_NO_LESS	Ignoring "Keep no less than"	The event is generated when a video fragment is deleted from the archive before expiration of the time period set by the Keep no less than parameter. See also Configuring video camera archive depth
AVAILABILITY	Availability of the key position in the license file and the number of objects	<p>The event occurs in response to the GET_AVAILABILITY command and contains information about the number of objects added to the license key for the given position. Parameters:</p> <ul style="list-style-type: none"> qty_localpriority<> is the number of allowed objects of the specified type on this computer (local key) qty<> is the total number of allowed objects of the specified type in the license (generic key) pos<> is the position number in the license key <p>If there is no position in the key, the event will have a zero value</p>
WRITING_FAILED	Writing error	
FILESYSTEM_FAILED	File system error	<p>Contains mandatory parameter error_msg<>—error text; can also contain parameters cam<>—camera ID and error_code<>—error code. Example:</p> <p>CAM -1 FILESYSTEM_FAILED error_msg<Failed to delete index file: D:\\VIDEO\\INDEX\\14061608.idx, error code: 5.>,error_code<5></p>

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) the parameters of a camera	rec_priority<>	Record priority (from 0 to 3, 0—standard, 3—all resources)
	compression<>	Compression ration (0—no compression, 1—maximum quality, ..., 5—minimum quality)
	sat_u<>	Value of color (0—min, 10—max)
	proc_time<>	Append period (0–30 sec)
	manual<>	Control brightness and contrast settings (0—manual; 1—auto; 2—auto, but close to values specified manually)

Command—command description	Parameters	Description of parameters
	contrast<>	Contrast (0—min, 10—max)
	md_size<>	Size of motion detection objects (1-16)
	md_mode<>	Mode of pause record (1—enabled, 0—disabled)
	audio_type<>	Type of sound accompaniment
	pre_rec_time<>	Time of pre-record (0-20 sec)
	bright<>	Brightness (0—min, 10—max)
	audio_id<>	Number of a microphone (empty parameter if there is no microphone)
	rec_time<>	Record speed (1-30 FPS, 0—not used)
	alarm_rec<>	Record of alarms (1—enabled, 0—disabled)
	hot_rec_time<>	Time of hot record (0-30 sec)
	hot_rec_period<>	Period of hot record (0-20 sec)
	mux<>	Number of channel (0-1 channel, 15-16 channel)
	color<>	Color (0—black and white, 1—multicolor)
	activity<>	-
	arch_days<>	Number of archive days
	blinding<>	Camera is sealed
	config_id<>	-
	decoder<>	-
	flags<>	Flags
	fps<>	Speed of record (0—not used, 1-30 FPS)
	ifreq<>	Frequency of key frames in sequence (1—each frame is key, 2—100th frame)
	mask 0, mask1, mask2, mask3, mask4	Detection tool mask
	md_contrast<>	Sensitivity of motion detection tool (0-15)

Command—command description	Parameters	Description of parameters
	motion<>	Estimation of compressor motion (5–255)
	name<>	Object name
	password_crc<>	Video archive password
	priority<>	Priority of record resource (0—auto, 1—manual)
	resolution<>	Resolution (0—standard CIF, 1—high 2CIF, 2—maximum 4CIF)
	type<>	Type of object
	yuv<>	Color schema of video signal coding (0—YUV4:2:0, 1—YUV4:2:2)
DELETE—disables camera	-	-
START_VIDEO—enables video stream for the current camera	slave_id<>	Name of the computer to which the camera is connected
	compress<>	Level of compression
	register_only<>	-
STOP_VIDEO—disables video stream for the current camera	slave_id<>	Name of computer to which camera is connected
REQUEST_MASK	mask<>	Mask
MUX1, MUX2, MUX3—display the image of a camera on 1, 2, 3 analog outputs	-	-
ACTIVATE—display camera on monitor	monitor<>	Number of a monitor
ARM—arm camera	-	-
DISARM—disarm camera	-	-
REC—start recording from a camera	time<>	Time of record in seconds, if null than only one frame is recorded
	rollback<>	If 1, the record is performed with a rollback
	priority<>	Sets priority of command to start recording. See Appendix 1. Priorities of start and stop recording commands
	stream_id<>	Sets an identification number of a stream for recording. The stream ID is set as "n.m" where n is the camera ID, m is the number of the stream. <i>Note. If the specified stream isn't used for any purpose other than record by command (and custom on clients), make sure that the Lock disabling streams not in use checkbox is set for it—see The</i>

Command—command description	Parameters	Description of parameters
		Settings panel of the Camera object
REC_STOP—stop recording from a camera	priority<>	Sets priority of a command to stop recording. See Appendix 1. Priorities of start and stop recording commands
	user_id<>	If recording was stopped by a user from the Video surveillance monitor, the parameter contains the user ID. Otherwise the parameter is absent
	from_macro<>	If recording was stopped by a macro, the parameter contains the macro ID. Otherwise the parameter is absent
SET_MASK—set mask	mask<>	Mask
ADD_SUBTITLES—add titles	command<>	Test of imposed titles
	title_id<>	The ID of the Captioner object which is used to impose
	page<>	Required parameter to allow recording titles to the titles database to provide search by titles. Available values: BEGIN (start of recording in the database), END (end of recording in the database)
SIP_CONNECT—Sip connected	-	-
SIP_DISCONNECT—Sip disconnected	-	-
SET_IPINT_PARAM—set (change) the parameters of an IP device. Reaction allows changing the IP device settings without going into its web-interface. <i>Note. For reaction operation it is required to enable the mode of multi-flow video signal—see Configuration of multistream video and Appendix 2. Defining param_id and param_value values for SET_IPINT_PARAM reaction</i>	param_id<>	Name of a 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 a 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 a video stream (optional parameter). It is given by “Number of camera”. “Number of a stream”, for example 1.1, 1.2
GET_FRAME—get frame from a camera even if it is not displayed in the Video surveillance monitor	path<>	Path to save a frame. If there is no parameter, the FRAME_SENT event with the data parameter will be formed in the system. Processing of this event is described in The SaveToFile method of the The Script object. Programming using the JScript language
	stream<>	Optional parameter. Sets <i>Intellect</i> stream to get a frame from. The stream can be specified by a number or purpose. Possible purposes: <ul style="list-style-type: none"> stream_archive—stream for archive recording stream_alarm—stream for archive recording by alarms stream_client—stream for displaying stream_analytic—stream for video analytics

Command—command description	Parameters	Description of parameters
		Stream number consists of a camera ID and a stream ID divided by dot, for example, 4.3 is for stream 3 from camera 4
	time<>	Optional parameter. It is set to request video frame from the archive. Format: DD-MM-YYYY hh:mm:ss. Example: time<19-09-2017 11:35:34>
	gate<>	Optional parameter. Specifies the network name of the Videogate from the archive of which to get the frame
	arch<>	Optional parameter. Specifies the network name of the Backup archive to get the frame from
	slave_id<>	Optional parameter. Specifies the network name of the Server to get the frame from
	password_crc<>	Optional parameter. Specifies the CRC sequence to be recorded to the file together with the frame
ARCH_DEL_RECORD—delete archive recordings over the specified period.	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 using which the bookmark will be created. Parameter format: <computer id>.<monitor id>. For example, slave_id<WS2.1>—WS2 is a computer ID and 1 is Video surveillance monitor ID
CRUISE_START—auto cruise	cruise_id<>	Route name on a camera
	action<>	Executed action: <ul style="list-style-type: none"> • CRUISE_START—start cruising along the specified route • PATROL_PLAY—start patrolling along the specified route

Command—command description	Parameters	Description of parameters
	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:\> <i>Note. The "\" character is an escape character.</i> The network path is set in the UNC format
	arch	Get the backup archive depth. Example. DoReactStr("CAM","2","GET_DEPTH","drive<D:\>,cam<2>,arch");
	gate	Get the videogate archive depth. Example. DoReactStr("CAM","1","GET_DEPTH","drive<V:\>,gate");
CLEAR_SUBTITLES—remove all titles from the video image	title_id<>	The ID of the Captioner object
GET_AVAILABILITY—check the availability of the key position	pos<>	The position number in the license key

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

Properties of the CAM object	Description of the object properties
ID<>	Object ID
PARENT_ID<>	Parent object ID
TELEMETRY_ID<>	Telemetry module ID (PTZ ID)
REGION_ID<>	Region ID

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

1. Switch camera to the colored mode and start recording from it when 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 when disabling the fifth video camera.

```
OnEvent("CAM","5","DETACH") // fifth video camera is disabled
```

```
{
  DoReact("CAM","1","ARM"); //first video camera is armed
}
```

- Use half of resources when 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
}
```

- Set maximal compression synchronously with the fourth microphone of audio card on the first video camera when 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<0LXA_LINE>, audio_id<4>"); //
  first video camera, maximal compression, synchronously with forth microphone of audio
  card.
}
```

- Start recording from the first camera with minimum 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.
}
```

- Start recording from the first camera in the “rollback” mode when it is 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
}
```

- Set new parameters of video signal when 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.
}
```

- 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 the **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

10.3 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 param<0> - additional information displayed in the corresponding column of the Event Viewer, in the following format: "ComputerName;ExportPeriod;UserName;UserID", for example, param0<LOCALHOST;04-06-18 16:50:55_04-06-18 16:55:55;Smith;1>

AVI_EXPORT_RESULT	Video export result	The event has the same parameters as START_AVI_EXPORT with additional error_result<> having one of the following values: 0 - export successful 1 - unknown 2 - busy 3 - not ready 4 - invalid interval 5 - file error
PLAY_START	-	Start the archive fragment playback
PLAY_STOP	-	Stop the archive fragment playback
INTERFACE_MANIPULATION	Visualization change	param<0> – additional information displayed in the corresponding column of the Event Viewer, contains the identifier of the camera that was moved around the layout
LAYOUT_DEL	Deleting layout	param<0> – additional information displayed in the corresponding column of the Event Viewer, contains the name of the deleted layout
LAYOUT_ADD	Adding layout	param<0> – additional information displayed in the corresponding column of the Event Viewer, contains the name of the added layout
LAYOUT_ACTIVATE	Changing active layout	param<0> – additional information displayed in the corresponding column of the Event Viewer, contains the name of the activated layout
REPLACE_CAM	Changing the camera position	param<0> – additional information displayed in the corresponding column of the Event Viewer, in the following format: <Camera 1 name> → <Camera 2 name>
ACTIVATE_CAM	Camera activated	auto_switch<> – indicates whether slide show (auto paging, auto scrolling) was enabled at the time of camera activation. This parameter can be used to turn off slide show when activating a Video surveillance window
CAM_EXPAND	The Video Surveillance Window expanded to the entire Monitor	The event is generated if the following registry keys are set (see Registry keys reference guide): <ul style="list-style-type: none"> • MaximizeCameraOnDbIClk=1 • MinimizeCameraOnDbIClk=1 Event parameters: param0<> – camera ID user_id<> – the ID of the user who performed the action
CAM_COLLAPSE	The Video Surveillance Window collapsed back	The event is generated if the following registry keys are set (see Registry keys reference guide): <ul style="list-style-type: none"> • MaximizeCameraOnDbIClk=1 • MinimizeCameraOnDbIClk=1 Event parameters: param0<> – camera ID user_id<> – the ID of the user who performed the action

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
	show<>	Optional parameter. Possible values: 0 – do not update the layout in the Monitor after removing the camera. There may be empty space not occupied by Video surveillance windows 1 – update the layout in the Monitor after removing the camera to minimize empty space
"REMOVE_ALL" – removes all cameras from monitor	-	-
"STOP_VIDEO" – stops video 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 <i>Note. See also PLACE_CAM_IN_LAYOUT_CELL</i>	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)
	gate_id<>	ID of the video gate to receive video through. The corresponding camera must be added and configured in this video gate - see Selecting and configuring the cameras for the Videogate module
	slave_id<>	ID of the computer to which the command is applied
"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<>	-

Command – command description	Parameters	Description
	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 – Backup archive 10 + Object ID External storage in the Monitor object settings panel (normally 11) – external storage
"SETUP" – sets parameters of monitor	no_update<>	-
	overlay<>	Disable the mode of speed-up displaying
	x<>	Coordinate of top-left corner (0 – 100)
	y<>	Coordinate of top-left corner (0 – 100)
	w<>	Size in horizontal direction (0 – 100)
	h<>	Size in vertical direction (0 – 100)
	max_cams<>	Maximum allowable number of cameras on the monitor
	min_cams<>	Minimum allowable number of cameras on the monitor
	compress<>	-
	panel<>	Show control panel (0 – disabled, 1 – enabled)
	panel_type<>	-
	s<>	-
	layout<>	-
	gate<>	-
	map_id<>	-
	enable<>	-
	topmost<>	1 – show screen always on top
	type<>	Type of Monitor object
allow_move<>	Allows moving of window	
arch_id<>	Archive ID	

Command – command description	Parameters	Description
	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
	gstream_verbosity<>	If the value is not set, the function for stream auto select is disabled If the value is minBPS , then the stream for displaying is selected automatically as described in Configuring an auto select of video stream for displaying
"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<>	-
	cam_id<>	The ID of the camera to the Video surveillance window of which the command is to be applied. If the identifier is not specified, the command is applied to the active Video surveillance window (see Active Surveillance window)
	key<>	Possible values: "ARCH_EDIT_DATE" – change date of search by archive; "ARCH_EDIT_TIME" – change time of search by archive; "ARCH_EDIT_ENTER" – enter changes of values in archive; "ARCH_EDIT_ESCAPE" – cancel editing of archive; "ARCH_EDIT_BACK"; "ARCH_EDIT_REPLACE"; "WINDOW_ZOOM_IN" – expand window of video surveillance; "WINDOW_ZOOM_OUT" – hide window of video surveillance; "ZOOM_IN" – image incoming; "ZOOM_OUT" – image removal; "CYCLE_REW" – scrolling 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;

Command – command description	Parameters	Description
		<p>"UP" – move the frame up in the Zoom mode;</p> <p>"DOWN" – move the frame down in the Zoom mode;</p> <p>"MODE_VIDEO" – video surveillance mode;</p> <p>"MODE_ARCH" – mode of archive video records playback;</p> <p>"MODE_ARCH2" – mode of archive video records playback 2;</p> <p>"MASK_SHOW" – show mask;</p> <p>"MASK_HIDE" – remove mask;</p> <p>"ARM" – arm camera;</p> <p>"DISARM" – disarm camera;</p> <p>"REW" – rewind;</p> <p>"PLAY" – play;</p> <p>"PLAY_NONSTOP" – non-stop playback;</p> <p>"PLAY_FAST" – speed up video record playback;</p> <p>"FF" – forward wind;</p> <p>"RECORD" – record;</p> <p>"RECORD_MIC" – record from microphone;</p> <p>"STOP" – stop;</p> <p>"REC_STOP" – stop record;</p> <p>"PAUSE" – pause;</p> <p>"MIC_ON" – microphone On;</p> <p>"MIC_OFF" – microphone Off;</p> <p>"PRINT" – print the frame;</p> <p>"SELECT_LAYOUT" – control layout of video surveillance monitor;</p> <p>"START_CYCLE_FF" – enable function of slide show of video surveillance windows frontwards. Period of switching video images is specified while configuring the Monitor interface object (see the Administrator's Guide, Configuring the display mode of camera windows);</p> <p>"STOP_CYCLE" – stop of slide show of Video surveillance windows;</p> <p>"EXPORT_DO" – open the AviExport tool for background export (see The AviExport utility);</p> <p>"PROTECT_DO" – open the dialog box to create a bookmark (see Create a bookmark);</p> <p>"PROTECT_VIEW" – show the bookmark list (see List of bookmarks)</p>
<p>"START_AVI_EXPORT" – starts video export</p> <p><i>Note. See the example as follows</i></p>	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	-	-

Command – command description	Parameters	Description
"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
"SHOW_LAYOUT" – show layout with the specified ID.	layout_id<>	Layout ID in database
"GO_LIVE" – switch all cameras on the monitor to live video mode.	-	-
"GO_ARCH" – switch all cameras on the monitor to archive mode.	arch_time<>	Optional parameter. Sets the time position in the archive in the format DD-MM-YY HH:MM:SS. By default, the archive is positioned on the last record
"SAVE_AS" – export selected archive fragment	-	-
PLACE_CAM_IN_LAYOUT_CELL – add camera to the specific cell on the specific layout on the Monitor	cam<>	ID of camera in the object tree which is to be displayed in the monitor. If the parameter value is incorrect, for example, 0 or -1, then the corresponding cell will be hidden
	layout_name<>	ID or name of the layout to add camera on
	cell<>	The number of the cell on the layout to which the camera should be added. Cells are numbered from top to bottom left to right, starting from the top-left corner of the layout. Important! Cells are numbered starting from 0. If some other camera has already been added to the specified cell, it will be replaced

Command – command description	Parameters	Description
SET_TITLES – show captions over a video image in any display mode. Such captions are not archived and are displayed until CLEAR_TITLES command is applied or Monitor is reset	cam<>	The ID of the camera to the Video surveillance window of which the command is to be applied
	titles<>	The caption text that should be displayed. Use '\r' to break the line
	title_id<>	Captioner ID
CLEAR_TITLES – disable the captions created with the help of the SET_TITLES command	cam<>	The ID of the camera to the Video surveillance window of which the command is to be applied
	title_id<>	Captioner ID

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

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

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 is finished.

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

OnEvent("CAM", "1", "MD_STOP");
```

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

4. 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 17:10:50 to the c:\aaa.avi file.

Examples of export starting in three ways: using the IIDK (port 900 and 1030) and using script:

- a. **IIDK (port 900)**

```
MONITOR|1|START_AVI_EXPORT|start<24-10-14 17:10:38>,finish<24-10-14 17:10:50>,avi_path<c:\aaa.avi>,cam<1>
```

- b. **IIDK (port 1030)**

```
CORE||DO_REACT|
```

```
source_type<MONITOR>,source_id<1>,action<START_AVI_EXPORT>,params<4>,param0_name<avi_path>,param0_val<c:\aaa.avi>,param1_name<cam>,param1_val<1>,param2_name<finish>,param2_val<24-10-14 17:10:50>,param3_name<start>,param3_val<24-10-14 17:10:38>
```

- c. **Script (start 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>");
}
```

7. When macro 1 is run, display the text

“NNN
Titles”

(with line break) over the video image of camera 1 using the captioner 1. When macro 2 is run, disable this text.

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

if (Event.SourceType == "MACRO" && Event.SourceId == "2" && Event.Action == "RUN")
```

```
{
DoReactStr("MONITOR","1","CLEAR_TITLES","cam<1>,title_id<1>");
}
```

10.4 PLAYER

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

Operator format to describe 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 back the audio file	file<>	Full path to the audio file in the .wav format (with the name of the file being played back. For example: C:\Program Files (x86)\Intellect\Wav\cam_alarm_1.wav)
	from_macro<ID>	Response flag of the audio file being played back. The response is sent from the macro (specifying the ID of the existing macro, ID > 0). <i>Note. The parameter isn't mandatory if the voice notification channel is configured in the Audio Player object interface (see Setting up the voice notification using Audio player object)</i>
"SETUP" – sets the audio player parameters	board<>	Sound unit of the archive player
	flags<>	Flags
	h<>	Height of settings dialog (0 – 100)
	name<>	Object name
	voice<>	Sound notification
	voice_board<>	Sound unit of notification
	w<>	Width of settings dialog (0 – 100)
	x<>	Left top corner of settings dialog (0 – 100)
	y<>	Left top corner of settings dialog (0 – 100)
"STOP_WAV" – stops audio file playback	-	-

Properties of the **PLAYER** object are given in the following table.

Properties of the PLAYER object	Description of properties
ID<>	Object ID

PARENT_ID<>	Parent object ID
-------------	------------------

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

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

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

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

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

10.5 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.

State of the OLXA_LINE object	State description
"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
}
```

10.6 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:

- 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");
}
```

- Close the operator query pane using macro 2.

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

10.7 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.

Command – command description	Parameters	Description
"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>");
}
```

10.8 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.

Command – command description	Parameters	Description
	attachments<>	Attachments. If several files are attached, their addresses are semicolon separated.
	flags<>	Flags.
	name<>	Object name.
	pack<>	Way of attachments packing.
	is_body_html<>	Specifies if HTML markup is to be applied when sending. Possible values: 1 or 0.
	inline<>	Specifies if attachments are only shown in the message text text (value of 1) or both in text and "Attachments" section (value of 0).
"SEND" – send mail message.	-	-
"SEND_RAW" – send e-mail with parameters	same as for SETUP	see Examples of scripts in JScript language

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

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

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<john.smith@axxonsoft.com>,
to<mary.foreman@axxonsoft.com>,attachments<"+ filename + ">");
    DoReact("MAIL_MESSAGE","1","SEND");
}

```

10.9 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>");
}
```

10.10 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");
}
```

10.11 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.	-	-

Command – command description	Parameters	Description
“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
}
```

10.12 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_" [,"_parameters_"]);
```

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.
"PLAY" – play audio file.	user<>	User name.
	file<>	Full path to the audio file in .wav format (indicating the name of the file being played. For example: C:\Program Files (x86)\Intellect\Wav\cam_alarm_1.wav). 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
"STOP" – stop playing audio file.	-	-
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. Play the audio file when video camera stops recording:

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

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

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

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

```
OnEvent("TIME_ZONE","1","ACTIVATE")
```

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

Note.

The **TIME_ZONE** object is described as follows (see [TIME_ZONE](#) section).

10.13 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>");
}
```

10.14 TELEMETRY

The **TELEMETRY** object corresponds to the **PTZ device** system object.

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

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

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

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

Event	Description	Comment
LOCKED	Locked	Event is received after the LOCK command (see the table below)
UNLOCKED	Unlocked	Event is received after the UNLOCK command (see the table below)

Operator format to describe actions with the PTZ devices 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 the end point of autopan	tel_prior<>	Priority (1—low, 2—medium, 3—high)
AUTOPAN_START—start autopan	tel_prior<>	Priority (1—low, 2—medium, 3—high)
AUTOPAN_START_P—specify the start point of autopan	tel_prior<>	Priority (1—low, 2—medium, 3—high)
AUTOPAN_STOP—stop autopan	tel_prior<>	Priority (1—low, 2—medium, 3—high)
CLEAR_PRESET—clear the selected preset	tel_prior<>	Priority (1—low, 2—medium, 3—high)
	preset<>	Preset
D2OFF—disable the additional dynamic settings for Panasonic PTZ video cameras used to increase the quality of analog video signal	tel_prior<>	Priority (1—low, 2—medium, 3—high)
D2ON—enable the additional dynamic settings for Panasonic PTZ video cameras used to increase the quality of analog video signal	tel_prior<>	Priority (1—low, 2—medium, 3—high)
DOWN—rotate video camera lens down	tel_prior<>	Priority (1—low, 2—medium, 3—high)
FOCUS_IN—zoom in	tel_prior<>	Priority (1—low, 2—medium, 3—high)
FOCUS_OUT—zoom out	tel_prior<>	Priority (1—low, 2—medium, 3—high)
FOCUS_STOP—stop zooming in/out of image	tel_prior<>	Priority (1—low, 2—medium, 3—high)
GO_PRESET—rotate video camera to the position specified on the preset	tel_prior<>	Priority (1—low, 2—medium, 3—high)
	preset<>	Preset
HOME—rotate video camera to the initial (home) position	tel_prior<>	Priority (1—low, 2—medium, 3—high)
IRIS_CLOSE—close diaphragm	tel_prior<>	Priority (1—low, 2—medium, 3—high)
IRIS_OPEN—open diaphragm	tel_prior<>	Priority (1—low, 2—medium, 3—high)
IRIS_STOP—stop diaphragm	tel_prior<>	Priority (1—low, 2—medium, 3—high)
LEFT—rotate video camera lens to the left	tel_prior<>	Priority (1—low, 2—medium, 3—high)
LEFT_DOWN—rotate video camera lens to the left and down	tel_prior<>	Priority (1—low, 2—medium, 3—high)
LEFT_UP—rotate video camera lens to the left and up	tel_prior<>	Priority (1—low, 2—medium, 3—high)

Command—command description	Parameters	Description
PATROL_LEARN—start procedure of patrol programming performed by recording the video camera actions	tel_prior<>	Priority (1—low, 2—medium, 3—high)
PATROL_PLAY—start patrolling	tel_prior<>	Priority (1—low, 2—medium, 3—high)
PATROL_STOP—stop patrolling	tel_prior<>	Priority (1—low, 2—medium, 3—high)
RIGHT—rotate video camera lens to the right	tel_prior<>	Priority (1—low, 2—medium, 3—high)
RIGHT_DOWN—rotate video camera lens to the right and down	tel_prior<>	Priority (1—low, 2—medium, 3—high)
RIGHT_UP—rotate video camera lens to the right and up	tel_prior<>	Priority (1—low, 2—medium, 3—high)
SET_PRESET—record the current position of video camera to the selected preset	tel_prior<>	Priority (1—low, 2—medium, 3—high)
	preset<>	Preset
STOP—stop video camera lens rotation	tel_prior<>	Priority (1—low, 2—medium, 3—high)
UP—rotate video camera lens up	tel_prior<>	Priority (1—low, 2—medium, 3—high)
SETUP—set up PTZ device	address<>	Device address
	cam<>	Camera ID to control
	flags<>	Flag of object operating (0—ON, 1—OFF)
	name<>	Object name of PTZ device
	speed<>	Speed
SEND_BUFFER—send command to COM port in the hexadecimal format	buffer<>	Command in the hexadecimal format
	parent_id<>	ID of Telemetry Controller parent object. The required parameter
	tel_prior<>	Priority (1—low, 2—medium, 3—high). The value of the parameter must be greater than 0
LOCK—lock. Switch the telemetry over to the LOCKED state for a specified time	tel_prior<>	Priority (1—low, 2—medium, 3—high). The value of the parameter must be greater than 0. It is forbidden to perform control commands with a lower priority than the specified during the lock time
	duration<>	Lock duration. If the parameter is not specified, the lock is valid until the UNLOCK command is executed
UNLOCK—unlock. Switch the telemetry over to the UNLOCKED state for a specified time	-	-

Command—command description	Parameters	Description
AUTOFOCUS_OFF—disable autofocus	tel_prior<>	Priority (1—low, 2—medium, 3—high). <input type="checkbox"/> To use this command, you must add it to the Reactions tab for the TELEMETRY object in ddi.exe (see The Reactions tab)

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

Properties of the TELEMETRY object	Description of the object properties
ID<>	Object ID of the PTZ device
PARENT_ID<>	Parent object ID

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

State of the TELEMETRY object	Description of the object state
LOCKED—locked	Control of telemetry is locked with some priority. It is forbidden to control telemetry with a priority lower than the specified when locking (see the table above)
UNLOCKED—unlocked	It is allowed to control telemetry with any priority

Examples of using 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 the position specified in the first preset with the relay enabled.

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

3. Record the patrol route for Camera 1 corresponding to the PTZ device 1.1. The route consists of two points, such that to go from point 1 to point 2, you need to rotate the camera to the left at speed of 6 for 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>");
}
    
```

10.15 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 <i>AXIS T8312</i> , 1 - <i>AXIS T8313 keyboard</i>
"KEY_RELEASED"	Key is released	param0<>	Code of released key	0..21 for <i>AXIS T8312</i> . For <i>BOSCH KBD-Digital</i> , <i>BOSCH KBD-Universal</i> and <i>Panasonic WV-CU950</i> see Installing and configuring security system components guide .
		device<>	Device on which key is released	0 – Main keyboard <i>AXIS T8312</i> , 1 – Rotary switch <i>AXIS T8313</i>
"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 <i>Panasonic WV-CU950</i> JogDial -1.. 1; Shuttle -6..6
		device<>	Type of used control mechanism <i>AXIS T8313</i>	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 <i>BOSCH KBD-Digital</i> or <i>BOSCH KBD-Universal</i> 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)

Command – command description	Parameters	Description
	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 <i>BOSCH KBD-Digital</i> or <i>BOSCH KBD-Universal telemetry panel</i>	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 <i>Panasonic WV-CU950 telemetry panel</i>	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

Command – command description	Parameters	Description
		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 <i>BOSCH KBD-Digital</i> or <i>BOSCH KBD-Universal telemetry panel</i> . Reaction without parameters for the <i>Panasonic WV-CU950 telemetry panel</i> .	display<>	0x00 – main display, 0x01 – status display
"RELE_ON" – turn on the light on the <i>AXIS T8312</i> keyboard or <i>Panasonic WV-CU950 panel</i>	rele<>	Code of key with the light, 12..16 for <i>AXIS T8312</i> . For <i>Panasonic WV-CU950</i> 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 <i>AXIS T8312</i> keyboard or <i>Panasonic WV-CU950 panel</i>	rele<>	Code of key with the light, 12..16
"RESET" – reset of <i>Panasonic WV-CU950 panel</i>	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 <i>Panasonic WV-CU950 panel</i>	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");
    }
}
```

10.16 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	Parameters	Parameter descriptions
"RUN"	Action is performed	src_sender<>	The name of the computer on which the macro was executed. <i>Note. The value of this parameter is displayed in the Add. info column of the Event viewer in real time. When Intellect is restarted and event log records are loaded from the database, this information is not displayed in the interface, but remains in the database.</i>
		user_id<>	The identifier of the user who executed the macro. <i>Note. The value of this parameter together with the user name is displayed in the Add. info column of the Event viewer in real time. When Intellect is restarted and event log records are loaded from the database, this information is not displayed in the interface, but remains in the database.</i>

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:

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");
}
```

10.17 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:

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>");
}
```

10.18 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:

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>")
}
```

10.19 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 re-indexing	
REBUILD_IN_STOP	End of archive re-indexing	
REGISTER_ATTEMPT	Attempt of unauthorized access	
REGISTER_ERROR	Limit of access attempts is exceeded	Event is generated when user failed to enter the system a lot of times. Some timeout is started after the event when user can't try to enter the system. Number of attempts and timeout can be changed using registry
REGISTER_USER	User registration	This event is generated when user tries to enter the system (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

Events	Description	Comment
SHUTDOWN	Shutdown	
DISC_MOUNT	Disc connected (mounted)	
DISC_UNMOUNT	Disc disconnected (unmounted)	
ARCHIVE_DEPTH	Archive depth	<p>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.</p> <p>Archive depth in format Days:Hours is specified in the Additional information field of the Event viewer while event displaying. Also this information contains in parameter of the param0<> event.</p> <p>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)</p>
FORCED_OFF	Forced offload	The event is generated before the forced unload of <i>Intellect</i> , for example, if the Guardant security key is removed. Unload is performed after the action caused it (for example, removing the Guardant key) after the time specified by the UnloadDelay registry key—see Registry keys reference guide
DEACTIVATE_ALL_DISP	Hide all displays	The event is generated when Hide all command is executed on a computer set in slave<> parameter. If except<> parameter is present, all displays except the one specified in this parameter are hidden
LIC_EXPIRATION	License expires in	<p>Not generated by default. Set NotifyExpireLic = 1 to enable (see Registry keys reference guide).</p> <p>The days<> parameter shows the number of days left till license expiration (can be non-integer). The event is generated at Intellect start-up and when the day changes</p>
DATABASE_ERROR	Database connection lost	The event is generated when the connection to SQL Server is lost the first time it is accessed after the disconnection
SCRIPT_ERROR	Script execution failed	By default, the event is not generated, since it is not added to the intellect.ddi external settings file. In order for the SCRIPT_ERROR event to be generated and added to the PROTOCOL table, it must be added to the intellect.ddi table (see Editing the external setting file (intellect.ddi) using the ddi.exe utility)

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

Command—command description	Parameters	Description
	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\ (HKEY_LOCAL_MACHINE \Software\Wow6432Node\ITV\INTELLECT registry section for 64-bits system	-	-

Command—command description	Parameters	Description
"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>:\<\> <i>Note. The "\" symbol is an escape character.</i> 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.

10.20 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
ACTIVATED	Display is activated on the remote computer. The computer name is transferred in the param0 <> parameter.

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 >");
}
```

10.21 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 occurs. Format of the 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 the list of working cameras corresponds to the list of the Videogate configuration
INACTIVE	Gate is inactive	Event is generated when there are no requests for video streams through the Videogate
ACTIVE_PART	Partial operation of gate	Event is generated when the number of working cameras is less than in the Videogate list

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

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

Operator format for operations with the Videogate:

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

The list of commands and parameters for the **GATE** object is presented in the table.

Command—command description	Parameters	Parameter description	Features
START_VIDEO—enable camera video stream and start writing to the archive	cam<>	Identifier of the camera by which it is necessary to start or stop recording	The commands work even if the Monitor doesn't display the selected camera.

Command—command description	Parameters	Parameter description	Features
STOP_VIDEO—stop camera video stream and stop writing to the archive			The commands work if constant recording and active camera recording is enabled in the Videogate—see Configuring the recording to the Videogate archive

10.22 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 occurs.

Format of the 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 type (initial state) 0—other 1—human or group of people (depending on the native_value<> parameter: if 1, human; if >1, group of people); 2—car 3—noise 4—object carried into the area 5—object carried out of the area
		native_value<>	To enable this parameter, set the following registry keys: VMDA.determineNoise, VMDA.determineGivenTaken, VMDA.determineHumanCar (see Registry keys reference guide). People counter for the "human" object type. Allows determining the quantity of people in the group. For other object types it is equal to -1
		param0<>	String value containing the event parameters from the VMDA detection
ALARM_END	End of alarm		
ARMED	VMDA detection is armed		
DISARMED	VMDA detection is disarmed		

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 when performing Macro 1:

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

Note

If cameras are connected using ONVIF Server, the events from the **VMDA detection** and other smart detection tools will be transferred as events from embedded detection tools—see [CAM_IP_DETECTOR](#).

10.23 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");
}
```

10.24 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 [The Script object. Programming using the JScript language](#).

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	<p>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:</p> <p>SET_MARKRECT – sends when face recognizing on the video image;</p> <p>DEL_MARKRECT – sends when face disappearing from the video image.</p> <p>Can be other parameters of the event which can be monitored using the Debug window (see The Script object. Programming using the JScript language, 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.</p> <p>For the DEL_MARKRECT value the camera number is forwarding in the param0_val parameter.</p>
SLAVE_CHAN GED	<p>Event is generated when Failover becomes active. It consists of the following parameters:</p> <p>old_slave_id – ID of the Computer object where cameras are moved from.</p> <p>new_slave_id – ID of the Computer object where cameras are moved to.</p> <p>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.</p>
CREATE_OBJ ECT	<p>Event triggers creation of an object. Parameters:</p> <p>objtype<> – object type, e.g. objtype<PERSON> for user creation.</p> <p>parent_id<> – identification number of the parent object.</p> <p>service_photo<> – when a user is created, the base64-encoded binary image of user photo can be put to this parameter. This is necessary for adding user photo immediately whet creating user in Access Manager.</p>

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+"&");
        ]
    }
}
```

10.25 IPJOYSTICK

The **IPJOYSTICK** object corresponds to the **Control device** object.

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

The format of the event procedure for the **IPJOYSTICK** object:

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

Description of the events of the **IPJOYSTICK** object.

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

10.26 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>");
}
```

10.27 MAP

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

The **MAP** object sends the events presented in the table. The procedure is started when the corresponding event occurs. The format of the events procedure for the Map:

<pre>OnEvent("MAP", "_id_", "_event_" [, "_parameters_"])</pre>	
Event	Description
LAYER_ACTIVATED	Layer activation. This event is received when a layer is selected on the Map. The obj_id<> parameter has the ID of the activated layer
ACTIVATE_OBJECT	<p>Object activation. The event is received when an object is selected (activated by mouse click) on the Map.</p> <p>Parameters:</p> <ol style="list-style-type: none"> 1. obj_type <>—object type 2. user_id<>—user ID 3. module<>—module name, for the Map—map.run 4. date<>—date when the event occurred 5. time<>—time when the event occurred 6. slave_id<>—computer network name 7. obj_id <>—object ID 8. layer<>—Map layer ID 9. fraction<>—millisecond when the event occurred 10. owner<>—user who activated the object 11. type_of_display <>—object display type, possible values: <ol style="list-style-type: none"> a. IMAGE—image b. IMAGE_AND_INDICATOR—image and indicator c. TEXT—text d. LINE—line e. POLYGON—polygon f. ELIPSIS—ellipse g. TITLE—the name of the object
OBJDBLCLK	The event is received when you double click an object on the Map. Contains the same parameters as ACTIVATE_OBJECT

Operator format to describe actions with the Map:

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


The list of commands and parameters for the **MAP** object is presented in the table.



Command	Parameters	Description
SET_TOPMOST —Set topmost	-	-
SET_NOTOPMOST —Cancel topmost	-	-
HIDE_OBJECT —Hide/show object icon on the map	objtype<>	Object type. Can be left 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 object ID is not set, then all objects of the specified type are hidden/shown
	hide<>	0—objects are shown on the map 1—objects are hidden on the map

SET_OBJECT_GEOMETRY — Set object location on the map	objtype<>	Object type
	objid<>	Object ID
	x<>	New coordinate of the top left corner of the object icon on the map layer along the X axis in pixels
	y<>	New coordinate of the top left corner of the object icon on the map layer along the Y axis in pixels
	exclude_children<>	By default, when using the SET_OBJECT_GEOMETRY reaction, when moving the object icons, the names of these objects (child objects) also move. If you pass the exclude_children <1> parameter in the reaction, then the object is moved separately from the children, that is, without their names
INSCRIBE —Inscribe to window	-	-
SHOW_MINIMAP —Show a minimap	x<>	The coordinate of the top left corner of the minimap along the X axis in pixels
	y<>	The coordinate of the top left corner of the minimap along the Y axis in pixels
	w<>	Width of the minimap in pixels
	h<>	Height of the minimap in pixels
	monitor<>	Monitor ID
	slave_id<>	Computer network name
SET_ZOOM —Change the Map scale	zoom<>	Map scale ratio
ACTIVATE_OBJECT — Activate object on the Map	obj_type<>	Object type
	obj_id<>	Object ID
	layer<>	Map layer ID. If the parameter is set, the script will work on the specified layer. If the parameter is not set, the scrip will work on the current layer
DRAW_ARROW —Draw a track of movement between objects	first_obj_type<>	Type of the object from which a track will be made
	first_obj_id<>	ID of the object from which a track will be made
	second_obj_type<>	Type of the object to which a track will be made
	second_obj_id<>	ID of the object to which a track will be made
	obj_id<>	ID of the created track

ERASE_ARROW —Erase a track of movement between objects	obj_id<>	ID of the track that should be erased. If you don't specify the parameter, all tracks will be erased
---	----------	--

Features of the **DRAW_ARROW** command execution:

1. When executing the command, a track will be displayed on each layer of the **Map** as arrows  between the objects.
2. If the objects are located on the same layer, the arrow is drawn directly between the specified objects. If the objects are located on different layers, the arrow is drawn by the shortest path.
3. You can limit the depth of searching for connections between layers to make a track with the DrawArrowSearchDepth key, see [Registry keys reference guide](#).
4. If
 - a. it isn't possible to make a track,
 - b. one of the objects doesn't exist,
 - c. it is possible to make a track, but the arrows cannot be displayed,

then the icon  will be displayed on the first object, and the icon  will be displayed on the second object.

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

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

10.28 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 [The Script object. Programming using the JScript language](#).

The operator format for describing the actions with the Failover is:

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

The list of parameters for the **FAILOVER** object is presented in the table below.

Command - command description	Comment
FORCED_START - forced transfer of the main Server configuration to the backup one.	The reverse configuration transfer is performed using the FORCED_STOP command or upon restarting/reconnecting the main Server.
FORCED_STOP - forced transfer of the backup Server configuration to the main one.	

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

OnEvent("OPERATORPROTOCOL","_id_", "_event_")		
Event	Event description	Event parameters
ACTIVATE_LEFT	Operator left-clicked the event cell in the Operator protocol	
ACTIVATE_RIGHT	Operator right-clicked the event cell in the Operator protocol	
POSTPONE_PRESSED	Operator clicked the Delay button	
CREATE_REPORT	Operator clicked the Create button on the Create report tab	The user_id<> parameter contains the user ID. The initial_date<> and final_date<> parameters specify the initial and final dates selected in the interface
RESPONSE_ALARM	Operator clicked the Alarm situation button	objtype<>—Object type objid<>—Object ID action<>—Name of the event in the database alarm_time<>—Time when the alarm occurred
RESPONSE_SUSPECT	Operator clicked the Suspicious situation button	
RESPONSE_FALSE	Operator clicked the False alarm button	
ACTIVATE_EVENT	Focusing on the event: click the event in the interface or jump to the required event using the keyboard	

The 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—command description	Parameters	Parameter description
DEL_ALARM—delete an 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 blank—delete all alarms
HIDE_BUTTON—hide the buttons used to assign status to event	button<>	Names of the buttons separated by comma: alarm—Alarm situation suspicious—Suspicious situation false—False alarm Example of setting a parameter: button<alarm,suspicious,false>
	hide<>	1—hide the buttons listed in the button parameter 0—show the buttons listed in the button parameter
	objtype<>	Object type
	objaction<>	Event type
	objid<>	Object ID

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

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

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

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

10.30 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

10.31 EVENT_VIEWER

The **EVENT_VIEWER** object corresponds to the **Event Viewer** system object.

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

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

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

Description of events of the **EVENT_VIEWER** object.

Events	Description
SHOW_ON_MAP	The operator runs the command "Display on the map"
SHOW_VIDEO	The operator runs the command "Display video"
SHOW_REPOR	The operator runs the command "Show report"
CREATE_REPORT	Generated if the GenerateEventInsteadOfReport registry key is set to 1 and the operator selects the "Show report" command. The report does not open. See also Registry keys reference guide .

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

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

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

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

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

10.32 CAM_TITLE

The **CAM_TITLE** object corresponds to the **Captioner** system object.

Operator format to describe actions with the Captioner is:

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

The list of commands and parameters for the **CAM_TITLE** object is given in the following table:

Command – command description	Comment
"REINDEX" – run titles database update	Titles database re-indexation is run at any value of the <code>_id_</code> parameter. See also The Cam_title_updater.exe utility to convert titles database.

Example.

Run titles database update on Macro 1.

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

10.33 CAM_FACECAPTURE

The **CAM_FACECAPTURE** object corresponds to the **Face Detection** system object.

The **CAM_FACECAPTURE** object sends the events presented in the table below. The procedure is started when the corresponding event appears. The event procedure format for the **CAM_FACECAPTURE** object:

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

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

Events	Events description
FACE_DETECTED	Face is captured
FACE_LEAVE	Face is lost

The operator format for describing the actions with the Face Detection is:

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

The list of parameters for the **CAM_FACECAPTURE** object is presented in the table below.

Parameters	Parameters description
owner	The name of the server where the face was captured/lost
fraction	The millisecond when the face was captured/lost
module	The module where the face is captured
date	The date when the face was captured/lost
guid_pk	The event ID (generated randomly for each event)
core_global	Distribute to all cores (notify all)

Parameters	Parameters description
guid	The captured/lost face ID (generated randomly for each event)
time	The time when the face was captured/lost
param0	Same as guid . Used to display the information in the "Add. info" column of the Event viewer

10.34 IPSTORAGE

The **IPSTORAGE** object corresponds to the **Edge storage** system object.

The operator format for describing the actions with the Failover is:

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

The list of parameters for the **IPSTORAGE** object is presented in the table below.

Command - description	Parameter	Parameter description	Comment
IMPORT – import missing Edge storage archive for the period	cam<>	Camera ID	The command can be applied if automatic import for the Edge storage was not successful for some reason. <i>Note. If import is performed at the time of sending the reaction, the command will not be executed. To abort the current import task, first send the UPDATE_TIME command.</i>
	datetime_from<>	Date and time to start import from in the following format: <DD-MM-YY HH:MM:SS>	
	datetime_to<>	Date and time to end import on in the following format: <DD-MM-YY HH:MM:SS>	
UPDATE_TIME – stop synchronization and set the time of the last import from external storage in the Settings.xml file	cam<>	Camera ID	The command is used to stop the current import task and execute the IMPORT command to synchronize the specified archive period.
	datetime<>	Date and time of the last synchronization to be set in the Settings.xml file	

Example.

Import archive from the camera 45 Edge storage for the period from 11-01-19 16:00:55 to 11-01-19 17:00:55 by Macro 10.

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

10.35 TELEGRAM

The **TELEGRAM** object corresponds to the **Telegram bot** system object.

The **TELEGRAM** object sends events presented in the table. The procedure is started when the corresponding event occurs. The format of the event procedure for a mail message.

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

Event	Description	Comment
ERROR	Message sending error	The error<> parameter contains the text description of the error

The format of the operator to describe actions with a mail message:

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

The list of commands and parameters for the **TELEGRAM** object is given in the table:

Command – command description	Parameters	Description
SEND – send	text<>	Message text
	chat_id<>	Chat identifier
	bot_id<>	Bot identifier
	longitude<>	Geolocation longitude
	latitude<>	Geolocation latitude
	address<>	Geolocation text address
SENDPHOTO – send photo	photo<>	Full path to the image file
	caption<>	File caption
	chat_id<>	Chat identifier
	bot_id<>	Bot identifier
	longitude<>	Geolocation longitude
	latitude<>	Geolocation latitude
	address<>	Geolocation text address

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

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

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

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

//Sending geolocation with chat ID and bot ID:
DoReact("TELEGRAM",1,"SEND","text<Hello
world>,chat_id<828752651>,bot_id<809045046:AAGtKxtDWu5teRGKW_Li8wFBQuJ-l4A9h38>","longitude<37.
3428359>,latitude<55.6841654>,address<Office>");
}
```

10.36 BACNET

The **BACNET** object corresponds to the **BacNet** system object.

The **BACNET** object generates events listed in the table below. Procedures start when the corresponding event occurs. The format of the event procedure for the **BacNet** object:

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

BACNET object events description:

Event	Description
ERROR	Error message received
EVENT_OCCURES	Message acknowledgment
WRITE_OCCURES	Recording confirmation
WRITE_RESULT	Recording result

Operator format for describing actions with the **BacNet** object :

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

The list of commands and parameters for the **BACNET** object is presented in the table.

Command - description	Parameters	Parameters description
WRITE - send value to BACnet device	bacnet_application_tag<>	Data type. Possible values: NULL = 0 BOOLEAN = 1 UNSIGNED INT = 2 SIGNED INT = 3 REAL = 4 DOUBLE = 5 OCTET STRING = 6 CHARACTER STRING = 7 BIT STRING = 8
	bacnet_value<>	Parameter value

	bacnet_objtype<>	Object type: ANALOG INPUT = 0 ANALOG OUTPUT = 1 ANALOG VALUE = 2 BINARY INPUT = 3 BINARY OUTPUT = 4 BINARY VALUE = 5
	bacnet_instance<>	BACnet unique global device identifier
	bacnet_property_id<>	Property id
	bacnet_device_id<>	BACnet device id in the system
EVENT - send message to BACnet device	event_type<>	Event type
	from_state<>	Change state from
	to_state<>	Change state to
	message_text<>	Event text

10.36.1 Examples

The examples code is given in JScript, see [The Script object. Programming using the JScript language.](#)

Write to an object using a script

```

var msg = CreateMsg();

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

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

//bacnet_property_id
var PROP_PRESENT_VALUE = 85;

```

```

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

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

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

DoReact(msg);

```

In case of successful script execution, an event will appear in the Debug window

Event :

```

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

```

Generate event

```

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

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

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

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

DoReact(msg);

```

If the module receives an event, the following event will be displayed in the Debug window:

Event :

```

BACNETINT|1|EVENT_OCCURES|
sender<Udp:47808>,slave_id<ASUS>,fraction<683>,owner<ASUS>,event_type<EVENT_CHANGE_OF_BITSTRING>,module<bacnet
int.vshost.exe>,message_text<test_text1!>,date<27-11-18>,guid_pk<{6D34BA08-1CF2-E811-8B83-C860008A29F9}>
>,from_state<EVENT_STATE_FAULT>,core_global<1>,adr<192.168.0.197:57878>,to_state<EVENT_STATE_NORMAL>,time<11:11:3
4>,source_guid<bd51a40d-1cf2-e811-8b83-c860008a29f9>

```

Read indicators from an object

```

var msg = CreateMsg();

//bacnet_application_tag
var BACNET_APPLICATION_TAG_NULL = 0;
var BACNET_APPLICATION_TAG_BOOLEAN = 1;
var BACNET_APPLICATION_TAG_UNSIGNED_INT = 2;
var BACNET_APPLICATION_TAG_SIGNED_INT = 3;
var BACNET_APPLICATION_TAG_REAL = 4;

```

```

var BACNET_APPLICATION_TAG_DOUBLE = 5;
var BACNET_APPLICATION_TAG_OCTET_STRING = 6;
var BACNET_APPLICATION_TAG_CHARACTER_STRING = 7;
var BACNET_APPLICATION_TAG_BIT_STRING = 8;

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

//bacnet_property_id
var PROP_PRESENT_VALUE = 85;

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

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

```

In case of successful reading, the following event will be displayed in the Debug window:

```

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

```

10.37 CAM_IP_DETECTOR

The **CAM_IP_DETECTOR** object corresponds to the **Embedded detector** system object.

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

Format of the event procedure for the **Embedded detector** object:

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

Description of events of the **CAM_IP_DETECTOR** object.

Events	Events description	Comment
DETECTED	Event	<p>The event occurs when metadata is received from an embedded detection tool. For example, this event occurs when receiving the data on body temperature from a thermal camera, etc.</p> <p>The event also occurs if the data from detection tools is transferred using Onvif.</p> <p>The param0<> parameter has a string value containing the event parameters. The transferred parameters depend on the embedded detection tool. If you use Onvif, you can configure the contents of the parameter on the Event settings tab of the ONVIF-Server object (see Filtering the ONVIF-Server events)</p>

Examples of events from the embedded detection tools:

Example 1

```
// Event from a thermal camera
Event : CAM_IP_DETECTOR|1|DETECTED|slave_id<QA-T51>,
fraction<16>,owner<QA-T51>,module<video.run>,date<23-04-20>,
guid_pk<{1345DC60-3485-EA11-8A95-B06EBF8119EF}>,core_global<1>,time<10:31:06>,
param0<TargetList:name=TargetList;type=6;TemperatureValue0:37.4;json0:{
  "BeginTime" : "20200423T073058.000000",
  "EndTime" : "20200423T073100.000000",
  "EventClass" : "FaceEvent",
  "Hypotheses" : [
    {
      "Age" : 0,
      "BestTime" : "20200423T073059.000000",
      "Gender" : "unknown",
      "Rectangle" : [ 0.6380, 0.550, 0.0680, 0.1560 ],
      "TemperatureValue" : 37.40
    }
  ],
  "Id" : 1
}
;>
```

Example 2

```
// Event from VMDA detection
Event:CAM_IP_DETECTOR|1|DETECTED|param0<Comment:ver_type<0>,objtype<SLAVE>,int_obj_id<1>,module
<video.run>,
core_global<1>,_TRANSPORT_ID<>,time<12:22:30>,objaction<PING>,onvif_event<>,
date<30-03-21>,slave_id<DESKTOP-JHRURJJ>,
objid<DESKTOP-JHRURJJ>;>,int_obj_id<1>,core_global<1>,
guid_pk<{9A989C70-3991-EB11-BDFF-00155DF96D00}>,slave_id<DESKTOP-339SH3U>,time<12:22:30>,_times
tamp<7520749>,
fraction<465>,date<30-03-21>,owner<DESKTOP-339SH3U>,module<video.run>
```

10.38 SIP_TERMINAL

The **SIP_TERMINAL** object corresponds to the **SIP-terminal** system object.

The **SIP_TERMINAL** object sends events given in the table. Procedure is started when the corresponding event occurs. Format of the event procedure for the **SIP-terminal** object:

```
OnEvent("SIP_TERMINAL","_id","_event")
```

Description of events of the **SIP_TERMINAL** object:

Event	Description	Contents of the param0<> parameter displayed in the Add. info field in the Event Viewer
CALL_END	Call end	Number of the subscriber who called

Event	Description	Contents of the param0<> parameter displayed in the Add. info field in the Event Viewer
CALL_END_OPERATOR	Operator call end	Numbers of the subscribers and call duration. For example, if the parameter takes the "903 to 906 (01:04)" value, it means that the subscriber 903 called the subscriber 906, and the call lasted 1 minute and 4 seconds
CALL_END_DEVICE	Device call end	
CALL_END_VIRTUAL	Special number call end	
CALL_BEGIN	Call start	Numbers of the subscribers: the subscriber who is calling and the subscriber who is being called
CALL_TRYING	Call attempt	
CALL_BEGIN_VIRTUAL	Start of special number call	
CALL_TRYING_VIRTUAL	Special number call attempt	

Operator format to describe actions with the SIP-terminal is as follows:

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

List of commands and parameters for the SIP_TERMINAL object:

Command—command description	Parameters	Parameters description
END_ALL_CALLS—end all calls on the specified terminal (regardless of whether the connection is established)	-	-

10.39 INC_MANAGER

The INC_MANAGER object corresponds to the **Incident manager** system object.

The INC_MANAGER object sends events given in the table. Procedures are started when the corresponding event occurs. Format of event procedure for the **Incident manager** object:

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

Events	Event description	Parameters	Parameter description	Comment
CLOSE_CLICK	Click the Close button in the interface			The event is generated when an incident is closed without being processed by the operator in the Incident manager interface
CLOSE_ALL_CLICK	Click the Close all button in the interface			The event is generated when all incidents are closed without being processed by the operator in the Incident manager interface

Events	Event description	Parameters	Parameter description	Comment
SELECT	Click an incident in the interface			The event is generated if the operator left-clicks or right-clicks on the incident in the Incident manager interface
ACTIVATE_EVENT	The operator selected the event (click the event with the mouse)	alarm_time<>	Time when the event occurred	
		event_guid<>	Event ID (generated randomly for each event)	
		objtype<>	Object type (e.g., CAM, GRELE, etc.)	
		action<>	Action type (e.g., MD_START, DISARM, etc.)	

10.40 INC_SERVER

The INC_SERVER object corresponds to the **Incident server** system object.

The INC_SERVER object sends events presented in the table. The procedure is started when the corresponding event occurs. The format of the event procedure for the **Incident server** object:

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

Events	Events description	Comment
EVENT	The event (incident) is taken into processing in the Incident manager or is being processed by the operator	<p>The event is generated:</p> <ol style="list-style-type: none"> 1) when the operator takes the event into processing; 2) at each step of the event processing. <p>The serializeBase64 parameter of the event contains JSON with the detailed information about the processed event, including the steps performed by the operator</p>

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

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

The list of commands and parameters for the INC_SERVER object is presented in the table.

Command—description of the command	Parameters	Parameters description	Comment
UPDATE_STATUS—change the status of the event (incident) in the Incident manager	pks<>	Array of event identifiers	Parameters are filters and the presence of at least one parameter is mandatory.
	objtypes<>	Objects types	Parameter values can be specified with a delimiter. This means that one OR the other value will be selected. Example: objjids<1 2>
	objjids<>	Objects identifiers	

Command—description of the command	Parameters	Parameters description	Comment
	actions<>	Actions	
	status<>	Event status: 0—Waiting to be processed 1—Processing 2—Suspended 3—Completed	
UPDATE_ESCALATE_STATUS—change the status of the event (incident) escalation in the Incident manager	escalated<>	Event escalation status: 0—Waiting to be processed (not escalated) 1—Escalated	
	pks<>, objtypes<>, objids<>, actions<> are the same as for UPDATE_STATUS		

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

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

Example 2. On macro 2, on the Incident server 2 change the escalation status of the camera 1 events to Waiting to be processed (not escalated)

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

11 Description of the object model in Intellect

11.1 The Core object and its built-in methods

11.1.1 The Core object

The **Core** object is a global static object providing the methods for monitoring and controlling the Intellect system objects. **Core** methods allow receiving information about the existing objects, generating reactions for them and changing their states. **Core** methods can pause script execution, script debugging, creating and calling global variables.

The **Core** object is not a prototype, thus no other objects can be created on its base (it cannot be used as a template). All **Core** methods are static. Thus, **Core** methods are called directly from the script with no need for a **Core** prefix.

11.1.2 The SetObjectParam method

The SetObjectParam method sets the values of object parameters.

Method call syntax

```
function SetObjectParam(objtype: String, id: String, param : String, value : String)
```

Method arguments:

1. **objtype** - required argument. The type of the object whose parameters are to be set. It takes the following values: Type – String, range – existing object types.
2. **id** - required argument. Identification number of the object of the type set in the objtype parameter. It takes the following values: Type – String, range – existing object identification numbers of the specified type.
3. **param** - required argument. The parameter of the object. It takes the following values: Type – String, range – available parameters of the object.
4. **value** - required argument. The value to be set for the parameter specified in the param argument. It takes the following values: Type – String, range – depends on the parameter.

Usage examples

Example. When Macro 1 starts, check if Cameras 1 to 4 are set to broadcast color video. If a camera is set for black-and-white video broadcast, then switch it to the color mode (setting the true ("1") value to the "Color" parameter – ("color")).

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
  var i;
  for(i=1; i<=4; i=i+1)
  {
    if (GetObjectParam("CAM", i , "color") == "0")
    {
      SetObjectParam("CAM", i, "color", "1");
    }
  }
}
```

Note

If the object is active when the script is started (i.e. the setting panel of this object is open), then object parameters can not be changed by the SetObjectParam method. For instance, if the setting panel for the Camera 1 object is open and the aforesaid script is started, the operation mode of Camera 1 will not be changed for the color one.

11.1.3 The SetObjectState method

The SetObjectState method changes the state of objects.

Method call syntax

```
function SetObjectState(objtype : String, id : String, state : String)
```

Method arguments:

1. **objtype** - Required argument. The type of the object whose state is to be changed. It takes the following values: Type – String, range – existing object types.
2. **id** - Required argument. Identification number of the object of the type set in the objtype parameter. It takes the following values: Type – String, range – existing object identification numbers of the specified type.
3. **state** - Required argument. The state to switch the object to. It takes the following values: Type – String, range – available states of the object.

Usage examples

Example. Check if Camera 1 is armed every hour. If Camera 1 is disarmed, arm it.

Note

The Timer object with identification number 1 should be created beforehand. Set the Minutes parameter of the Timer object to 30. The timer would go off every hour at half past the hour - 09:30, 10:30, 11:30, etc.

```
if (Event.SourceType == "TIMER" && Event.SourceId == "1" && Event.Action == "TRIGGER")
{
    if (GetObjectState("CAM", "1") == "DISARMED")
    {
        SetObjectState("CAM", "1", "ARMED");
    }
}
```

11.1.4 The DebugLogString method

The DebugLogString method outputs the user messages into the debug windows of the Editor-Debugger utility.

Method call syntax

```
function DebugLogString(output : String)
```

Method arguments:

output - Required argument. The text message to be displayed in the debug window of the Editor-Debugger utility. It takes the following values: Type – String.

Usage examples

Problem. Output to the debugger window all microphone events registered by the system.

```
if (Event.SourceType == "OLXA_LINE")
{
    var msgstr = Event.MsgToString();
    DebugLogString("Event from the microphone " + msgstr);
}
```

11.1.5 The Base64Decode method

The Base64Decode method is used for decoding the lines that are coded by Base64 scheme.

Method call syntax

```
function Base64Decode(data_in: String, WideChar: Boolean)
```

Method arguments:

1. **data_in** - required argument. Set a line that should be decoded in Base64;
2. **WideChar** - required argument. Determines coding type. Can take 0 or 1 values. If coding type is Unicode, argument value is 1, otherwise 0.

Usage examples

Decode the line that is set in Base64 by starting macro №1. Output the decoding result into the debug windows of the Editor-Debugger utility. (Result is « Intellect JAVA SCRIPT» line).

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
    var str = Base64Decode("SW50ZWxsZWN0IEpTY3JpcHQ= ", 0);
    DebugLogString(str);
}
```

11.1.6 The Sleep method

The Sleep method pauses the execution of the script for a specified period of time.

Method call syntax

```
function Sleep(milliseconds : int)
```

Method arguments:

milliseconds - Required argument. The length of time that the script will be inactive for. Set in milliseconds. It takes the following values: Type – int.

Usage examples

Problem 1. When Macro 1 starts, play the following audio files one by one: cam_alarm_1.wav, cam_alarm_2.wav, cam_alarm_3.wav from the ...\\Intellect\\Wav\\ folder. Set a 5 seconds (5000 milliseconds) delay before starting each subsequent file.

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
    var i;
    for(i=1; i<=3; i=i+1)
    {
        DoReactStr("PLAYER", "1", "PLAY_WAV", " file<\\cam_alarm_" + i + ".wav>");
        Sleep(5000);
    }
}
```

Problem 2. When macro №2 starts, timer №1, that triggers every 10 seconds in a minute after macro №2 starting, starts.

Note

To start this script create the **Timer** object with ID=1 beforehand. Leave object parameters set by default (**Any**). The **Timer 1** object can be disabled

```

if (Event.SourceType == "MACRO" && Event.SourceId == "2" && Event.Action == "RUN")
{
for(i=0; i<=5; i=i+1)
{
DoReactStr("TIMER","1", "DISABLE", "");
Sleep(10000);
DoReactStr("TIMER","1", "ENABLE", "");
NotifyEventStr("TIMER","1", "TRIGGER", "");
}
DoReactStr("TIMER","1", "DISABLE", "");
}

```

11.1.7 The Itv_var method

The **Itv_var** method sets and returns the values of global variables.

Method call syntax

```
function Itv_var (globalvar : String) : String
```

Globalvar – required argument. The name of the global variable. It takes the following values: Type – String, satisfying the rules for the names of the string parameters in the Windows registry.

Global variables are stored in the Windows registry to maintain their values after Windows restart. All global variables are stored in the registry branch HKEY_USERS\S-1-5-21-...\Software\VMSScript\VMSSCRIPT and HKEY_CURRENT_USER\Software\VMSScript\VMSSCRIPT. To access a global variable directly from the registry, search the registry for it by its name.

Usage examples

Problem. When Macro 1 starts, save the value of the bright parameter of Camera 10 to the cam10bright global variable. When Macro 2 starts, set the bright parameter of Cameras 1 to 4 to the value of the cam10bright global variable.

```

if (Event.SourceType == "MACRO" && Event.Action == "RUN")
{
  if(Event.SourceId == "1")
  {
    Itv_var("cam10bright") = GetObjectParam("CAM", "10", "bright");
  }
  if (Event.SourceId == "2")
  {
    var cam10bright = Itv_var("cam10bright");
    for(i=1; i<=4; i=i+1)
    {
      SetObjectParam("CAM", i, "bright", cam10bright);
    }
  }
}

```

11.1.8 The Int_var method

The **Int_var** method sets and returns values of global variables of integer type.

⚠ Attention!

The `Int_var` method uses the same storage as [the `Int_var` method](#), but modify the type of variable to the integer type.

Method call syntax:

```
function Int_var (globalvar : String) : int
```

Globalvar – required argument. The name of the global variable. It takes the following values: type – String, satisfying the rules for the names of the string parameters in the Windows registry.

📄 Note.

Global variables are stored in the system registry to maintain their values after Windows restart. All global variables are stored in the registry branch `HKEY_USERS\S-1-5-21-...\Software\VMSScript\VMSSCRIPT` and `HKEY_CURRENT_USER\Software\VMSScript\VMSSCRIPT`. To access a global variable directly from the registry, search the registry for it by the same name.

Usage examples. To check the method operation in the following test example the value 1 sets to the "2" global variable and then increases per 1 and displays on the script debug window.

```
if(Event.Action == "RUN")
{
    Int_var(2) = 1;
    Int_var("2")++;
    DebugLogString(Int_var("2").toString());
}
```

11.1.9 The GetObjectParentType method

The `GetObjectParentType` method returns the type of the parent object of the current object according to the object hierarchy.

Method call syntax

```
function GetObjectParentType (objtype : String) : String
```

Method arguments:

objtype - Required argument. The type of the object whose parent's type should be returned. It takes the following values: Type – String, range – existing object types.

The Main object is the highest level object in the hierarchy. It is the parent for all objects of the Computer, Screen, and other types.

Usage examples

Problem. When Macro 1 starts, display in the debugger window the names of four object types of higher hierarchical order starting from the detection zone.

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
    var objtype = "CAM_ZONE";
    DebugLogString(objtype);
    for(var i = 1; i<=4; i=i+1)
    {
```

```

    objtype = GetObjectParentType(objtype);
    DebugLogString(objtype);
}
}

```

11.1.10 The GetIPAddress method

The GetIPAddress method returns the IP-address of the Intellect kernel according to current video surveillance system architecture.

Method call syntax

```
function GetIPAddress (dst : String, src : String) : String
```

Method arguments:

1. **dst** - Required argument. The name of the remote computer where the Intellect kernel is installed. The value of dst should correspond to one of the names of the computers registered during setup of the video surveillance system. It takes the following values: Type – String, meeting the requirements for network computer names; range – computer names existing in the system.
2. **src** - Required argument. The name of the local computer where the script executes. The value of src should match the name of the local computer as it is registered in Intellect. It takes the following values: Type – String; meeting the requirements for network computer names.

The information about all connections of the local computer (kernel) to other remote computers (kernels) registered during the setup of the distributed architecture, is displayed in the **Architecture** tab of the **System Settings** window.

Usage examples

Problem. Upon a camera alarm, determine the name of the server this camera is connected to, and output the IP-address of the connection between this server and the local computer where the script executes, to the debugger window.

```

if (Event.SourceType == "CAM" && Event.Action == "MD_START")
{
    var camid = Event.SourceId;
    var compname = GetObjectParentId("CAM", camid, "COMPUTER");
    var ip = GetIPAddress("WS1","WS1"); \\ if the script is run on the computer where kernel of
Intellect software has been installed
    DebugLogString("IP-address of the alarmed camera computer" + ip);
}

```

11.1.11 The GetObjectName method

The GetObjectName method returns the name of the object that it was given upon creation.

Method call syntax

```
function GetObjectName(objtype : String, id : String) : String
```

Method arguments:

1. **objtype** - Required argument. The type of the object whose name is to be returned. It takes the following values: Type – String, range – existing object types.
2. **id** - Required argument. Identification number of the object of the type set in the objtype parameter. It takes the following values: Type – String, range – existing object identification numbers of the specified type.

Usage examples

Problem. In case of alarm in any sensor, open the information window with the following text - "Alarm in the <alarmed sensor name> sensor connected to the <server name which the sensor is connected to> server".

Note

Create the information dialog window beforehand using the Arpedit.exe utility and save it as test.dlg in the <Intellect>\Program folder.

```
if (Event.SourceType == "GRAY" && Event.Action == "ALARM")
{
    var grayid = Event.SourceId;
    var grayname = GetObjectName("GRAY", grayid);
    var compname = GetObjectParentId("GRAY", grayid, "COMPUTER");
    DoReactStr("DIALOG", "test", "CLOSE_ALL","");
    DoReactStr("DIALOG", "test", "RUN","Alarm in the '" + grayname + "' sensor connected to the '" + compname + "' server.");
}
```

11.1.12 The GetObjectState method

The GetObjectState method returns the state of the object at the moment of method call.

Method call syntax

```
function GetObjectState(objtype : String, id : String) : String
```

Method arguments:

1. **objtype** - Required argument. The type of the object whose state is to be returned. It takes the following values: Type - String, range - existing object types.
2. **id** - Required argument. Identification number of the object of the type set in the objtype argument. It takes the following values: Type - String, range - existing identification numbers of the objects of the specified type.

Usage examples

Problem. When Relay 1 activates (for example, on pressing the button connected to Relay 1), arm Sensor 1. The next time Relay 1 activates, disarm Sensor 1.

```
if (Event.SourceType == "GRELE" && Event.SourceId == "1" && Event.Action == "ON")
{
    if(GetObjectState("GRAY", "1")== "DISARM")
    {
        SetObjectState("GRAY", "1", "ARM");
    }
    else
    {
        SetObjectState("GRAY", "1", "DISARM");
    }
}
```

Note

Some object types can have several states at the same time. For example: ATTACHED|DISSOLVED or ADDED|OFF|RECORDER_ON|RECORD

11.1.13 The GetObjectParam method

The GetObjectParam method returns the value of the specified parameter of the object at the moment of method call.

Method call syntax

```
function GetObjectParam(objtype : String, id : String, param : String) : String
```

Method arguments:

1. **objtype** - Required argument. The type of the object whose parameter's value is to be returned. It takes the following values: Type – String, range – existing object types.
2. **id** - Required argument. Identification number of the object of the type set in the objtype argument. It takes the following values: Type – String, range – existing identification numbers of the objects of the specified type.
3. **param** - Required argument. The name of the parameter whose value is to be returned. It takes the following values: Type – String, range – available parameters of the object.

Usage examples

See the example for the SetObjectParam method.

11.1.14 The GetObjectParentId method

The GetObjectParentId method returns the identification number of the parent of the specified object.

Method call syntax

```
function GetObjectParentId(objtype : String, id : String, parent : String) : String
```

Method arguments:

1. **objtype** - Required argument. The type of the object whose parent's identification number should be returned. It takes the following values: Type – String, range – existing object types.
2. **id** - Required argument. Identification number of the object of the type set in the objtype argument. It takes the following values: Type – String, range – existing identification numbers of the objects of the specified type.
3. **parent** - Required argument. The type of the object which is the parent of the object type specified by the objtype argument. It takes the following values: Type – String, range – existing object types.

Usage examples

Problem. If a camera turns off or stops transmitting a video signal, send an e-mail message with the following subject: "Warning! Camera turned off" and, in the message body, the number of the camera and of the server it is connected to.



Note

The Short Messages Service is supposed to be installed and working properly

```
if (Event.SourceType == "CAM" && Event.Action == "DETACH")
{
    var cam_id = Event.SourceId;
    var parent_comp_id = GetObjectParentId("CAM", cam_id, "SLAVE");
    DoReactStr("MAIL_MESSAGE", "1", "SETUP", "from<***@mail.com>,to<***@mail.com>,body<Camera
disabling "+cam_id+" on the Server"+parent_comp_id+">,parent_id<1>,subject<Attention! Camera
disabling>,name<Message 1>,objname<Message 1>");
    DoReactStr("MAIL_MESSAGE", "1", "SEND", "");
}
```

11.1.15 The DoReactStr method

The DoReactStr method generates the response actions for the objects. It sends the action to the specified object. The action is transferred directly to the kernel where the object belongs, and not to the whole system. The action is specified as a group of String arguments.

Method call syntax

```
function DoReactStr(objtype : String, id : String, action : String, param<value> [,
param<value>] : String)
```

Method arguments:

1. **objtype** - Required argument. The type of the object that the action should be generated for. It takes the following values: Type – String, range – existing object types.
2. **id** - Required argument. Identification number of the object of the type set in the objtype argument. It takes the following values: Type – String, range – existing identification numbers of the objects of the specified type.
3. **action** - Required argument. The action to be generated. It takes the following values: Type – String, range – available actions for the objects of the specified type.
4. **param<value>** - Required argument. Several arguments of this type can be specified. The parameters of the action.

One parameter has the following syntax:

"param<value>", where

param – name of the parameter;

value – value of the parameter.

Several parameters have the following syntax:

"param1<value1>,param2<value2>..."

Elements of the list are separated by commas with no white space. If no parameters need to be specified, an empty string is used:

```
DoReactStr("CAM","1","REC","");
```

The param argument can take the following values: Type – String, range – available parameters of the specified action. The value argument can take the following values: Type – String, range – depends on the parameter.

For all reactions it is possible to specify delay of reaction performing using delay<> parameter. Delay is specified in seconds.

Note

Two types of system messages are available in the Intellect system: events and actions.

The events usually contain some information and are used as notifications sent to all Intellect kernels connected to each other during the system setup.

The actions are the control commands sent to specific objects. An action is transmitted only to the kernel where the related object belongs, and not to the whole system.

The [DoReactStr](#) and [DoReact](#) methods are used to generate actions. The [NotifyEventStr](#) and [NotifyEvent](#) methods are used to generate events.

Usage examples

Problem. When an alarm is received from a camera, switch Monitor 1 to single window mode and show the video from the alarmed camera in this window.

```
if (Event.SourceType == "CAM" && Event.Action == "MD_START")
{
  var camid = Event.SourceId;
  DoReactStr("MONITOR","1","ACTIVATE_CAM","cam<"+ camid +">");
  DoReactStr("MONITOR","1","KEY_PRESSED","key<SCREEN.1>");
}
```

```
}

```

Problem. When alarm by some camera is completed the record is to be continued for 5 second and after this time the record will be stopped (analogue of Post-record mode).

```
if (Event.SourceType == "CAM" && Event.Action == "MD_STOP")
{
    var camid = Event.SourceId;
    DoReactStr("CAM",camid,"REC_STOP","delay<5>");
}

```

Problem. Use macros 1 to enable telemetry control using mouse on the camera 4 displayed in the monitor 10. Use macros 2 to disable it.

```
if (Event.SourceType == "MACRO" && Event.Action == "RUN" && Event.SourceId == "1")
{
    DoReactStr("MONITOR","10","CONTROL_TELEMETRY","cam<4>,on<1>");
}
if (Event.SourceType == "MACRO" && Event.Action == "RUN" && Event.SourceId == "2")
{
    DoReactStr("MONITOR","10","CONTROL_TELEMETRY","cam<4>,on<0>");
}

```

11.1.16 The DoReact method

The DoReact method generates actions for the objects. It sends the action to the specified object. The action is transferred directly to the kernel where the object belongs, and not to the whole system. The action is specified using the MsgObject object.

Method call syntax

```
function DoReact(msgevent : MsgObject)

```

Method arguments:

msgevent - Required argument. The action sent to the specified object. It takes the following values: MsgObject objects created earlier in the script.

Note

Two types of system messages are available in the Intellect system: events and actions.

The events usually contain some information and are used as notifications sent to all Intellect kernels connected to each other during the system setup.

The actions are the control commands sent to specific objects. An action is transmitted only to the kernel where the related object belongs, and not to the whole system.

The [DoReactStr](#) and [DoReact](#) methods are used to generate actions. The [NotifyEventStr](#) and [NotifyEvent](#) methods are used to generate events.

Usage examples

Problem. When Relay 1 closes, close Relays 2 and 3. When Relay 1 opens, open Relay 2.

```
if (Event.SourceType == "GRELE" && Event.SourceId == "1")
{
    var msgevent = Event.Clone();
    if(Event.Action == "ON")
    {

```

```

    msgevent.SourceId = "2";
    DoReact(msgevent);
    msgevent.SourceId = "3";
    DoReact(msgevent);
}
if(Event.Action == "OFF")
{
    msgevent.SourceId = "2";
    DoReact(msgevent);
}
}

```

11.1.17 The DoReactSetupCore method

The DoReactSetupCore method is used for changing the parameters of the object. It changes only the specified parameters, leaving other parameters intact.

Method call syntax

```

function DoReactSetupCore(objtype : String, id : String, param<value> [, param<value>] :
String )

```

Method arguments

1. **objtype** - Required argument. The type of the object whose parameters are to be changed. It takes the following values: Type – String, range – existing object types.
2. **id** - Required argument. Identification number of the object of the type set in the objtype argument. It takes the following values: Type – String, range – existing identification numbers of the object of the specified type.
3. **param<value>** - Required argument. Several arguments of this type can be specified. The parameters of the action.

One parameter has the following syntax:

“param<value>”, where

param – name of the parameter;

value – value of the parameter.

Several parameters have the following syntax:

“param1<value1>,param2<value2>...”.

Elements of the list are separated by commas with no white space.

The param argument can take the following values: Type – String, range – available parameters of the specified action. The value argument can take the following values: Type – String, range – depends on the parameter.

Usage examples

Problem. When Macro 1 starts, set the values of the following parameters of Cameras №1-4: PTZ device number (telemetry_id) and synchronous microphone number (audio_id). The values should be equal to the camera numbers plus 1.

```

if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
    var i;
    for(i=1; i<=4; i=i+1)
    {
        DoReactSetupCore ("CAM", i, "telemetry_id<" + (i+1) + ">,audio_id<" + (i+1) + ">");
    }
}

```

11.1.18 The DoReactSetup method

The DoReactSetup method is used for temporary changing parameters of the object. It changes only the specified parameters, leaving other parameters intact.

Method call syntax

```
function DoReactSetup (objtype : String, id : String, param<value> [, param<value>] :
String )
```

Method arguments:

1. **objtype** - required argument. The type of the object the parameters of which are to be changed. It takes the following values: type – String, range – existing object types.
2. **id** - required argument. Identification number of the object of the type set in the objtype argument. It takes the following values: type – String, range – existing identification numbers of the objects of the specified type.
3. **param<value>** - required argument. Several arguments of this type can be specified. The parameters of the action.

One parameter has the following syntax:

“param<value>”, where

param – name of the parameter;

value – value of the parameter.

Several parameters have the following syntax:

“param1<value1>,param2<value2>...”.

Elements of the list are separated by commas with no white space.

The param argument can take the following values: values of the String type, range is limited by available parameters of the specified action. The “value” argument can take the following values: values of the String type, the range depends on the parameter.

Example. When Macro 1 starts, temporary remove all cameras on the first monitor.

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
DoReactSetup ("MONITOR","1","REMOVE_ALL","");
}
```

11.1.19 The DoReactGlobal method

The DoReactGlobal method is used to generate object reactions. The DoReactGlobal method transmits a reaction to the required object. The reaction is transmitted not only to the kernel, where the object is registered, but to the whole system. For the DoReactGlobal method the reaction is specified by the MsgObject object.

Method call syntax

```
function DoReactGlobal(msgevent : MsgObject)
```

Method arguments:

msgevent - required argument. It sets a reaction transmitted to the required object. It takes the following values: MsgObject objects created earlier in the script.

Example. When macro 2 starts, guard on Sensor 2. The command is to be transmitted to all system kernels as the reaction to be registered in the Events Log.

```

if (Event.SourceType == "MACRO"&& Event.SourceId == "2" && Event.Action == "RUN")
{
var msgevent = CreateMsg();
msgevent.SourceType = "GRAY";
msgevent.SourceId = "2";
msgevent.Action = "ARM";
DoReactGlobal(msgevent);
}

```

11.1.20 The NotifyEventStr method

The NotifyEventStr method generates system events. Events are sent to all kernels connected to the local kernel. An event is specified as a group of String arguments.

Method call syntax

```

function NotifyEventStr(objtype : String, id : String, event : String, param<value> [,
param<value>] : String )

```

Method arguments:

1. **objtype** - Required argument. The type of the object that the event should be generated for. It takes the following values: Type – String, range – existing object types.
2. **id** - Required argument. Identification number of the object of the type set in the objtype argument. It takes the following values: Type – String, range – existing identification numbers of the object of the specified type.
3. **event** - Required argument. The event to be generated. It takes the following values: Type – String, range – available events for the objects of the specified type.
4. **param<value>** - Required argument. Several arguments of this type can be specified. The parameters of the event.

One parameter has the following syntax:

"param<value>", where

param – name of the parameter;

value – value of the parameter.

Several parameters have the following syntax:

"param1<value1>,param2<value2>...".

Elements of the list are separated by commas with no white space. If no parameters need to be specified, an empty string is used:

```

DoReactStr("CAM", "1", "MD_START", "");

```

The param argument can take the following values: Type – String, range – available parameters of the event. The value argument can take the following values: Type – String, range – depends on the parameter.

Note

Two types of system messages are available in the Intellect system: events and actions.

The events usually contain some information and are used as notifications sent to all Intellect kernels connected to each other during the system setup.

The actions are the control commands sent to specific objects. An action is transmitted only to the kernel where the related object belongs, and not to the whole system.

The [DoReactStr](#) and [DoReact](#) methods are used to generate actions. The [NotifyEventStr](#) and [NotifyEvent](#) methods are used to generate events.

Usage examples

Problem. When an alarm is received, send the “panic lock” event corresponding to the camera region. For camera identification numbers from 1 to 4, use region 1; for camera numbers from 5 to 10, use region 2.

```

if (Event.SourceType == "CAM" && Event.Action == "MD_START")
{
    var regionid;
    if (Event.SourceId <=4)
    {
        regionid = "1";
    }
    if ((Event.SourceId > 4) && (Event.SourceId <= 10))
    {
        regionid = "2";
    }
    NotifyEventStr("REGION", regionid, "PANIC_LOCK", "");
}

```

11.1.21 The NotifyEvent method

The NotifyEvent method generates system events. The event is sent to all kernels connected to the local kernel. The event is specified using the MsgObject object.

Method call syntax

```
function NotifyEvent(msgevent : MsgObject)
```

Method arguments:

msgevent - Required argument. The event sent to the system. It takes the following values: MsgObject objects created earlier in the script.

Note

Two types of system messages are available in Intellect system: events and reactions.

The events usually contain some information and are used as notifications sent to all Intellect kernels connected to each other during the system configuration.

The reactions are the control commands sent to specific objects.

The reactions are transmitted only to the kernel where the object belongs, and not to the whole system.

The [DoReactStr](#) and [DoReact](#) methods are used to generate reactions. The [NotifyEventStr](#) and [NotifyEvent](#) methods are used to generate events.

Example. When the Backup Archive 1 module starts archiving video recordings, the analog output 1 of the Video Capture Device 2 is disabled. Send the command as an event to be registered in the Events Log.

Note

While executing this script, the analog output 1 of the Video Capture Device 2 is not disabled

```

if (Event.SourceType == "ARCH" && Event.SourceId == "1" && Event.Action == "ACTIVE ")
{
    var msgevent = CreateMsg();
    msgevent.SourceType = " GRABBER ";
    msgevent.SourceId = "2";
    msgevent.Action = "MUX1_OFF";
    NotifyEvent(msgevent);
}

```

```
}

```

11.1.22 The NotifyEventGlobal method

The NotifyEventGlobal method is used to generate system events. The generated event is transmitted to all system kernels connected via the net. For the NotifyEventGlobal method the event is specified using the MsgObject object.

Method call syntax

```
function NotifyEventGlobal (msgevent : MsgObject)
```

Method arguments:

msgevent - required argument. It sets the event transmitted to the system. It takes the following values: MsgObject objects created earlier in the script.

Example. When Macro 1 starts, the first camera is set for recording. The command is to be transmitted to all system kernels as the event to be registered in the Events Log.

Note

While executing this script, camera1 is not set for recording.

```
if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
var msgevent = CreateMsg();
msgevent.SourceType = "CAM";
msgevent.SourceId = "1";
msgevent.Action = "REC";
NotifyEventGlobal(msgevent);
}
```

11.1.23 The CreateMsg method

The CreateMsg method creates objects based on the MsgObject prototype.

Method call syntax

```
function CreateMsg() : MsgObject
```

Method arguments: no arguments.

Usage examples

Problem 1. When an alarm is received, send the “panic lock” event corresponding to the camera region. For camera identification numbers from 1 to 4, use region 1, for camera numbers from 5 to 10, use region 2.

```
if (Event.SourceType == "CAM" && Event.Action == "MD_START")
{
var msgevent = CreateMsg();
msgevent.SourceType = "REGION";
msgevent.Action = "PANIC_LOCK";
if (Event.SourceId <=4)
{
msgevent.SourceId = "1";
}
```

```

}
if ((Event.SourceId > 4) && (Event.SourceId < 10))
{
    msgevent.SourceId = "2";
}
NotifyEvent(msgevent);
}

```

Problem 2. When timer №1 starts, start macro №1 every 30 seconds.

Note

To start this script create the **Timer** object with ID=1 beforehand. Set value=1 to the **Second** parameter of the **Timer** object, leave other parameters without changing («**Any** by default)

```

if (Event.SourceType == "TIMER" && Event.SourceId == "1" && Event.Action == "TRIGGER")
{
    var msg = CreateMsg();
    msg.StringToMsg(GetObjectParams("TIMER", "1"));
    if(msg.GetParam("s") == "1")
    {
        DoReactStr("MACRO", "1", "RUN", "");
        SetObjectParam("TIMER", "1", "s", "30");
        DoReactStr("TIMER", "1", "DISABLE", "");
        DoReactStr("TIMER", "1", "ENABLE", "");
    }
    if(msg.GetParam("s") == "30")
    {
        DoReactStr("MACRO", "1", "RUN", "");
        SetObjectParam("TIMER", "1", "s", "1");
        DoReactStr("TIMER", "1", "DISABLE", "");
        DoReactStr("TIMER", "1", "ENABLE", "");
    }
}
}

```

11.1.24 The Lock and Unlock methods

The Lock and Unlock methods are used to create a global critical section when synchronization of scripts started in different streams is required. The Lock method opens a critical section and the Unlock method closes it.

Attention!

Attention! If the Lock method has been called, then the Unlock method has to be called too. Otherwise the system can freeze.

It is recommended to avoid using the Lock and Unlock methods.

Method call syntax

```
function Lock()
```

```
function Unlock()
```

Example. When Macro 1 starts, calculate total alarmed relays and sensors. Objects of each type are to be calculated at the same time (in an individual script). The result is to be stored to “counter” global variable.

Script 1:

```
// Number of alarmed relays is calculated
var i = Number(0);
if (Event.SourceType == "MACRO" && Event.SourceId== "1" && Event.Action == "RUN")
{
var msg = CreateMsg();
msg.StringToMsg(GetObjectIds("GRELE"));
var objCount = msg.GetParam("id.count");
var k;
for(k= 0; k < objCount; k++)
if(GetObjectState("GRELE", msg.GetParam("id." + k))= "ALARM"){
    Lock();
    i = Itv_var("counter");
i++;
Itv_var("counter")=i;
Unlock();
}
}
```

Script 2:

```
//Number of alarmed sensors is calculated
var i = Number(0);
if (Event.SourceType == "MACRO" && Event.SourceId== "1" && Event.Action == "RUN")
{
var msg = CreateMsg();
msg.StringToMsg(GetObjectIds("GRAY"));
var objCount = msg.GetParam("id.count");
var k;
for(k = 0; k < objCount; k++)
if(GetObjectState("GRAY", msg.GetParam("id." + k))= "ALARMED"){
    Lock();
    i = Itv_var("counter");
i++;
Itv_var("counter")=i;
Unlock();
}
}
```

Note

If the Lock() and Unlock() methods are not used in this example, then collisions may occur and calculated value will be less than a real one

11.1.25 The IsAvailableObject method

The IsAvailableObject method is used to determine the current access rights to an object.

Method call syntax:

```
function IsAvailableObject(compname: String, objtype: String, id: String, param : String) :
String
```

The method returns 0 if the current user has not been assigned access rights of type **param** for the object; it returns 1 if these rights have been assigned.

Method arguments:

1. **compname** - required. Corresponds to the name of the **Computer** object on which the object was created in the hardware tree.
2. **objtype** - required. Corresponds to the type of the system object to which access rights are being checked. Allowed values: String type; the set of values is restricted to the object types registered in the system.
3. **id** - required. Corresponds to the identification (registration) number of the object of the type specified by the **objtype** argument. Allowed values: String type; the set of values is restricted to the identification numbers of objects of the specified type, which are registered in the system.
4. **param** - required. Corresponds to the number of the type of rights which are being checked. A description of rights is given in [Limiting access to the system objects administration, control and viewing functions](#) of the [Administrator's Guide](#).

Allowed values:

- a. 0 - NoView access rights. The method will return 1 if there are no administrative, control-, or viewing rights for the object (red 'x').
- b. 1 - NoControl access rights. The method will return 1 if there are only viewing rights (letter M).
- c. 2 - ViewAndControl access rights. The method will return 1 if there are control- and viewing rights for the object (green box).
- d. 3 - ViewOrControl access rights. The method will return 1 if there is either viewing or control rights for the object.
- e. 4 - Not access rights (e.g. no access rights).
- f. 5 - Configure access rights. The method will return 1 if there are administrative, control- and viewing rights for the object (gray box).

For example: Based on a **Computer** object named "S-UYUTOVA", a **Camera** object with the identifier 1 has been created in the hardware tree. To determine the current access rights to the object:

```
var i = 0;
for(i = 0; i <= 5; i++)
{
    var result =
    IsAvailableObject('S-UYUTOVA', 'CAM', '1', i);
    DebugLogString("right "+i+" = "+result);
}
```

11.1.26 The GetUserId method

The GetUserId method returns the identifier of the current *Intellect* Software System user.

Method call syntax:

```
function GetUserId (cmp : String) : String
```

Method arguments:

1. **cmp** - required. Specifies the name of the computer on which the *Intellect* Software System is installed. Allowed values: values of String type that satisfy the requirements for computers' network names; the set of values is restricted to the names of computers registered in the system.

For example: To display the identifier of the current user of the *Intellect* Software System, which has been installed on a computer named 'WS3':

```
DebugLogString(GetUserId("WS3"));
```

11.1.27 The GetEventDescription method

The GetEventDescription method is in use for receiving the object description on free language.

Syntax of method invocation:

```
function GetEventDescription (obj_type : String, event : String)
```

Method arguments:

1. **obj_type** – required argument. Specifies type of system object description of which is required to get.
2. **event** – required argument. Specifies the name of object description of which is required to get.

Example. Display messages about receiving of events for camera 1 on the free language in the Debug window.

```
if (Event.SourceType == "CAM"&& Event.SourceId == "1")
{
    var str = GetEventDescription("CAM", Event.Action);
    DebugLogString(str);
}
```

11.1.28 The GetObjectIdByParam method

The GetObjectIdByParam method allows receiving of object ID at which some parameter is equal to the specified value. The first found object ID is receiving if there are several such objects. The null is receiving if such objects are not found

Syntax of method invocation:

```
function GetObjectIdByParam (obj_type : String, obj_param : String, param_value : String)
```

Method arguments:

1. **obj_type** – required argument. Specifies type of system object, ID of which is required to get.
2. **obj_param** - required argument. Specifies the name of parameter in database on value of which the object is to be found.
3. **param_value** – required argument. Specifies the required value of object parameter.

Example. Find cameras from which black-and-white image is receiving and set the **Color** parameter equals to 1 for them.

```
if (Event.SourceType == "MACRO" && Event.SourceId== "1" && Event.Action == "RUN")
{
    var id = GetObjectIdByParam("CAM","color","0"); //receiving the first object ID
    while (id){ //while exist the Camera objects from which black-and-white image is receiving
        SetObjectParam ("CAM", id, "color", "1"); //change the Color parameter for found object
        id = GetObjectIdByParam("CAM","color","0"); //receiving the next object ID
    }
}
```

```
}

```

11.1.29 The SaveToFile method

SaveToFile method is in use to save in file the frame from camera receiving in the data parameter of FRAME_SENT event.

Syntax of method invocation:

```
function SaveToFile (path: String, data: String, param : Boolean)
```

Saving of frame is also can be performed using the GET_FRAME reaction of the CAM object. It is required to specify the path for saving file with frame in the path parameter of this reaction. FRAME_SENT event is created in the system if the GET_FRAME reaction has not got the path parameter. In the data parameter of the FRAME_SENT event the video image frame which is to be saved using the SaveToFile method is stored.

The reaction allows exporting of video image frame even if the camera is not displaying in the Monitor window.

Method argument:

1. **path** – mandatory argument. Specifies the full path to save the file with frame.
2. **data** – mandatory argument. Specifies data to save in file.
3. **param** – mandatory argument. Defines necessity of conversion from base 64 format before saving. Possible parameter values:
 - a. true – decode from base64 before saving;
 - b. false – save string without conversion.

Time of frame saving depends on the anchor frames frequency. The higher the anchor frames frequency, the less time.

Example. Save the frame receiving from Camera 1 in the test.jpg file on the D disk:

```
if (Event.SourceType == "CAM" && Event.SourceId == "1" && Event.Action == "FRAME_SENT")
{
  SaveToFile("D:\\test.jpg",Event.GetParam("data"),true);
}
```

11.1.30 The GetLinkedObjects method

The GetLinkedObjects method is used to get list of objects linked to the specified camera using the Objects connection object (see the Administrator's Guide, Connection of objects with cameras section)

Method call syntax:

```
function GetLinkedObjects(type1 : string, id : string, type2 : string)
```

Method argument:

1. **type1** – the type of object for which linked objects are to be returned.
2. **id** – identification number of object for which linked objects are to be returned.
3. **type2** – the type of linked objects which are to be returned. If empty string is sent then linked objects of all types will be returned.

Example.

The **Objects connection** object is configured the following way:

Display in debug window the list of objects linked with the camera 1.

```
if (Event.SourceType == "MACRO")
```

```
{
varmsgstr = GetLinkedObjects("CAM","1","MACRO")

DebugLogString("Linked objects " + msgstr);
}
```

As a result the "Linked objects MACRO:1" message will be displayed in the script debug window.

11.1.31 The WriteIni method

The WriteIni method is used for writing the string variable to ini-file.

Method call syntax:

```
function WriteIni(varName: String, varValue: String, path: String)
```

Method argument:

1. **varName** – required argument. Sets name of variable for saving in the file.
2. **varValue** – required argument. Sets value of t variable.
3. **path** – required argument. Sets full path to the ini-file in which variable is to be stored. Storage of variables can be replaced on the network resource. Enter the network path in the argument for it.

Example. Write the MyVar variable to the \\fs\temp\test.ini file and specify the «Helloworld!» value to it. Then read the written value and display it on the script debug window.

```
WriteIni("MyVar", "Hello world", "\\fs\temp\test.ini");

var result = ReadIni("MyVar","\\fs\temp\test.ini");

DebugLogString(result);
```

11.1.32 The ReadIni method

The ReadIni method is used to read values of string variable in the ini-file.

Method call syntax:

```
function ReadIni (varName: String, path: String): String
```

Method call syntax:

1. **varName** – required argument. Sets name of the variable storing in the file.
2. **path** – required argument. Sets the full path to the ini-file in which variable is storing.

See example in [The WriteIni method](#) section.

11.1.33 The AddIni method

The AddIni method is used to write, change and read integer variable from the ini-file. The method returns the value of variable after its changing.

Method call syntax:

```
function AddIni(varName: String, varValue: int, path: String): int
```

1. varName – required argument. Sets the name of variable in the file.
2. varValue – required argument. Sets the value of variable or value which should be added to the existing value of variable:
 - a. The varValue value will be assigned to the variable if there is a file with the varName name and string value in the file.
 - b. If there is no variable with the varName name in the file then such variable will be created and the varValue value will be assigned to it.
 - c. If there is a variable with the varName name in the file and it has integer value or its value is indicated to the integer type, then value will be indicated and the varValue value will be added to it.
3. path – required argument. Sets the full path to the ini-file in which variable is to be stored. Storage of variables can be placed on the network resource. Enter the network path for it.

Example. There is no the "MyVar" variable in the "C:\\test.ini". Write such variable with the -1 value to the file, then add 1 to it and display the result value on the script debug window.

```
var result = AddIni("MyVar", -1, "C:\\test.ini");

result = AddIni("MyVar", 1, "C:\\test.ini");

DebugLogString(result);
```

11.1.34 The SetTimer method

The SetTimer method is used to start the timer.

The syntax for accessing the method is:

```
function SetTimer (id : int, milliseconds : int)
```

Method arguments:

1. **id** is a required argument. It specifies the timer ID. Allowed values are int or string type.
2. **milliseconds** is a required argument. It specifies the period with which the timer will trigger if it is not stopped by [the KillTimer method](#). It is specified in milliseconds. Allowed values are int type.

Example. 2 seconds after the macro 1 is executed, start recording on camera 1.

```
if(Event.SourceType=="LOCAL_TIMER" && Event.Action=="TRIGGERED" && Event.SourceId==333) //it is possible to use Event.SourceId == "333", i.e. string ID
{
  var actuallyKilled = KillTimer(333);
  if(actuallyKilled == 1)
  {
    DoReactStr("CAM","1","REC","");
  }
}

if(Event.SourceType=="MACRO"&& Event.SourceId == "1" && Event.Action == "RUN")
{
  SetTimer(333,2000); //333 - id, 2000 msec = 2 sec - period
}
```

11.1.35 The KillTimer method

The KillTimer method is used to stop the timer. Returns 1 if timer was stopped as a result of function executed.

The syntax for accessing the method is:

```
function KillTimer (id : int) : int
```

Method arguments:

1. **id** is a required argument. It specifies the timer ID. Allowed values are int or string type.

Example. See in the [The SetTimer method](#) section.

11.1.36 The GetObjectChildIds method

The GetObjectChildIds method returns IDs of child objects of specified type created under given object.

Method call syntax

```
function GetObjectParentId(parent : String, id : String, objtype : String) : String[]
```

Method arguments:

1. **parent** - Required argument. The type of the object for which you want to find out child objects . It takes the following values: Type – String, range – existing object types.
2. **id** - Required argument. Identification number of the object of the type set in the **parent** argument. It takes the following values: Type – String, range – existing identification numbers of the objects of the specified type.
3. **objtype** - Required argument. The type of the child objects whose identification numbers should be returned. It takes the following values: Type – String, range – existing object types.

Example. Arm all cameras on WS2 computer on Macro 1.

```
if (Event.SourceType == "MACRO" && Event.Action == "RUN" && Event.SourceId == "1")
{
    var children = GetObjectChildIds("SLAVE", "DESKTOP-UBOS6BK", "CAM");
    ch=children.split(",");
    for (i=0;i<ch.length; i++ )
    {
        DoReactStr("CAM",ch[i],"ARM","");
    }
}
```

11.1.37 The Base64EncodeFile method

The Base64EncodeFile method encodes a file to Base64. The method returns a string.

See also [The Base64Decode method](#) and [The SaveToFile method](#).

Method call syntax:

```
function Base64EncodeFile (data_in: String): String
```

Method arguments:

1. **data_in** – required argument. Sets a path to the file to be encoded.

Example. At running Macro 1, encode 1.bmp to Base64 and save it to 2.bmp.

```

if (Event.SourceType == "MACRO" && Event.SourceId == "1" && Event.Action == "RUN")
{
    var s = Base64EncodeFile("d:\\1.bmp");
    SaveToFile("d:\\2.bmp",s, true);
}

```

11.1.38 The Base64EncodeW method

The Base64EncodeW method encodes a Unicode string to Base64. The method returns a string.

See also [The Base64Decode method](#).

Method call syntax:

```

function Base64EncodeW (data_in: String): String

```

Method arguments:

1. **data_in** – required argument. Sets a string to be encoded.

Example. Decode a Unicode string to Base64 then encode back and put to the debug log.

```

var test =
Base64Decode("MAAZAC0AMAA3AC0AMgAwADEAOQA6ADQAMA6AA0ACgB0AGUAcwB0ACAAMQANAAoAM
AAZAC0AMAA3AC0AMgAwADEAOQA6ADQAMgA6ADIAMQA6AA0ACgB0AGUAcwB0ACAAMgA=", true);
DebugLogString("----->" + test);
var res = Base64EncodeW(test);
DebugLogString("----->" + res);

```

If the Base64Decode method received the "true" parameter, the script will have the following output:

```

03-07-2019 15:39:40:
test 1
03-07-2019 15:42:21:
test 2

```

11.1.39 The run_cmd and run_cmd_timeout methods

The run_cmd method is intended to execute commands on the command prompt from a script.

The run_cmd_timeout method is intended to execute commands on the command prompt with the process termination timeout.

When invoking commands, the command prompt window does not open, i.e. the commands are executed in silent mode.

The syntax for calling methods is as follows:

```

function run_cmd (cmd: String)
function run_cmd_timeout (cmd: String, timeout: int)

```

Method arguments:

1. **cmd** is a command prompt command.
2. **timeout** (only for run_cmd_timeout) is a timeout for process to terminate after the command execution.

Example 1. Run the curl utility and send a POST request with the text "Hello" to the test URL <https://postman-echo.com/post>

```

var s = run_cmd("curl --request POST --url https://postman-echo.com/post --data \"Hello\"");

```

```
DebugLogString(s);
```

Example 2. Show CPU usage on [Graph 1](#) updating on [Timer 3](#).

```
var id = "1"; // id of the Charts object
var timer_id = "3"; // id of the Timer object to trigger the script
slave_id = "DESKTOP-5397BVV"; // id of the Computer object

if (Event.SourceType == "TIMER" && Event.Action == "TRIGGER" && Event.SourceId == timer_id)
{
    var date = Event.GetParam("date");
    var time = Event.GetParam("time");
    var cpu = "for /f \"tokens=2* delims=^, \" %k in ('typeperf \\Processor
Information(_Total)\\% Processor Time\" -sc 1 ^| findstr \":\")( do echo %k";
    var cpu_usage = run_cmd(cpu);
    var cpu_usage2 = cpu_usage.replace(/\"/g, "");
    var cpu_usage3 = cpu_usage2.replace(/\"/g, "");
    DebugLogString(cpu_usage3);
    DoReactStr("ANALOGCHART", id, "ANALOG_PARAMS", "int_obj_id<"+id+">,parent_id<>,slave_id<"+slav
e_id+">,objid<"+id+">,chan<5>,core_global<1>,text<"+cpu_usage3+">,
min_val<0>,max_val<100>,sensor_id<cpu_usage>,time<"+time+">,date<"+date+">");
}
```

11.1.40 The WriteIniAny method

The WriteIniAny method is used for writing a string variable to an ini-file. Unlike the WriteIni method, in WriteIniAny you can specify the required file section for writing.

Method call syntax:

```
function WriteIniAny(varName: String, varValue: String, path: String, section: String)
```

Method arguments:

1. varName — required argument. Sets the name of the variable for saving in the file.
2. varValue — required argument. Sets the value of a variable.
3. path — required argument. Sets the full path to the ini-file where the variable should be stored. The storage of variables can be placed on a network resource; for this, specify the network path in this argument.
4. section — required argument. Sets the name of the section of the ini-file where the variable should be stored.

Example. Write the MyVar variable to the "config" section of the C:\\Backup\\test2.ini file, and specify the "Hello world!" value to it. Then read the written value and display it in the debug window of the script.

```
WriteIniAny("MyVar", "Hello world!", "C:\\Backup\\test2.ini", "config");
var result = ReadIniAny("MyVar", "C:\\Backup\\test2.ini", "config");
DebugLogString(result);
```

11.1.41 The ReadIniAny method

The ReadIniAny method is used to read the values of a string variable in the ini-file. Unlike the ReadIni method, in ReadIniAny you can specify the required file section from which you want to read the variable.

Method call syntax:

```
function ReadIniAny (varName: String, path: String, section: String): String
```

Method arguments:

1. `varName` — required argument. Sets the name of the variable saved in the file.
2. `path` — required argument. Sets the full path to the ini-file where the variable is stored.
3. `section` — required argument. Sets the name of the section of the ini-file from which you want to read the variable.

See example in [The WriteIniAny Method](#) section.

11.1.42 The AddIniAny method

The `AddIniAny` method is used to write, change and read integer variable from the ini-file. Unlike the `AddIni` method, in `AddIniAny` method you can specify the section of the file that contains the integer variable. The method returns the value of the variable after its changing.

Method call syntax:

```
function AddIniAny(varName: String, varValue: int, path: String, section: String): int
```

1. `varName` – required argument. Sets the name of variable in the file.
2. `varValue` – required argument. Sets the value of variable or value which should be added to the existing value of variable:
 - a. The `varValue` value will be assigned to the variable if there is a variable with the `varName` name and string value in the file.
 - b. If there is no variable with the `varName` name in the file then such variable will be created and the `varValue` value will be assigned to it.
 - c. If there is a variable with the `varName` name in the file and it has integer value or its value is indicated to the integer type, then value will be indicated and the `varValue` value will be added to it.
3. `path` – required argument. Sets the full path to the ini-file in which variable is to be stored. Storage of variables can be placed on the network resource. Enter the network path for it.
4. `section` — required argument. Sets the name of the section of the ini-file where the variable is stored.

Example. There is no "MyVar" variable in the "config" section of the "C:\\test.ini" file. Write such variable with the -1 value to the file, then add 1 to it and display the result value on the script debug window.

```
var result = AddIniAny("MyVar", -1, "C:\\test.ini", "config");

result = AddIniAny("MyVar", 1, "C:\\test.ini", "config");

DebugLogString(result);
```

11.2 The MsgObject and Event objects and their built-in methods and properties

11.2.1 The MsgObject and Event objects

MsgObject is a prototype (template) used to create objects with methods and properties for handling events. The methods and properties of **MsgObject** allow receiving the information about objects that send or receive events, generating actions, changing object states, etc.

The methods and properties of the **MsgObject** prototype can be called via its instance objects or the **Event** static object.

Event is a static object used to call events in Intellect. The **Event** object represents the system event that launched the script. All **MsgObject** methods and properties are available for the **Event** object.

The `CreateMsg` method of the **Core** object is used to create instances based on the **MsgObject** prototype.

11.2.2 The GetSourceType method

The GetSourceType method returns the type of the **MsgObject** or **Event** object.

Method call syntax

```
function GetSourceType() : String
```

Method arguments: no arguments.

Usage examples

Problem. When Macro 1 starts, arm Detection Zones *.1 in the Day mode for Cameras № 1 – 4. When Macro 2 starts, arm Detection Zones *.2 in the Night mode for Cameras № 1 – 4. When Macro 3 starts, arm Detection Zones *.3 in the Rain mode for Cameras № 1 – 4.

Note

Symbol "*" corresponds to identification number of a camera in the system (from 1 to 4)

```
if(Event.GetSourceType() == "MACRO" && Event.GetAction() == "RUN")
{
  var k;
  //Switching the cameras to the Day mode by arming the *.1 detection zones
  if(Event.GetSourceId() == "1")
  {
    for (k = 1; k<= 4; k = k+1)
    {
      DoReactStr("CAM_ZONE", k + ".1", "ARM", "");
      DoReactStr("CAM_ZONE", k + ".2", "DISARM", "");
      DoReactStr("CAM_ZONE", k + ".3", "DISARM", "");
    }
  }

  //Switching the cameras to the Nigh mode by arming the *.2 detection zones
  if(Event.GetSourceId() == "2")
  {
    for (k = 1; k<= 4; k = k+1)
    {
      DoReactStr("CAM_ZONE", k + ".1", "DISARM", "");
      DoReactStr("CAM_ZONE", k + ".2", "ARM", "");
      DoReactStr("CAM_ZONE", k + ".3", "DISARM", "");
    }
  }

  //Switching the cameras to the Rain mode by arming the *.3 detection zones
  if(Event.GetSourceId() == "3")
  {
    for (k = 1; k<= 4; k = k+1)
    {
      DoReactStr("CAM_ZONE", k + ".1", "DISARM", "");
      DoReactStr("CAM_ZONE", k + ".2", "DISARM", "");
      DoReactStr("CAM_ZONE", k + ".3", "ARM", "");
    }
  }
}
```

11.2.3 The GetSourceId method

The GetSourceId method returns the identification number of the **MsgObject** or **Event object**.

Method call syntax

```
function GetSourceId() : String
```

Method arguments: no arguments.

Usage examples

See the example for the GetSourceType method.

11.2.4 The GetAction method

The GetAction method returns the event received as an Event object or specified for a **MsgObject** object.

Method call syntax

```
function GetAction() : String
```

Method arguments: no arguments

Usage examples

See the example for the GetSourceType method.

11.2.5 The GetParam method

The GetParam method returns the value of the specified parameter of the **MsgObject** or **Event** object.

Method call syntax

```
function GetParam(param: String) : String
```

Method arguments:

param - required argument. The name of the parameter of the object created using **MsgObject** (or of the **Event** object). It takes the following values: type – String, range – available parameters for the objects of the specified type.

Note

If the object has no parameter with this name, then the method restores an empty string

Example. When any event from any camera is registered, check from which computer the event comes. If the computer has “WS3” name, create the event copy where the computer’s name is “Computer”.

```
if (Event.SourceType == "CAM")
{
var msg = Event.Clone();
if (msg.GetParam("slave_id") == "WS3")
{
msg.SetParam("slave_id", "Computer");
NotifyEvent(msg);
}
}
```

11.2.6 The SetParam method

The SetParam method assigns a value to the specified parameter of the **MsgObject** or **Event** object. It changes only the specified parameters, leaving other parameters intact.

Method call syntax

```
function SetParam(param : String, value : String)
```

Method arguments:

1. **param** - Required argument. The name of the parameter of the object created using **MsgObject** (or of the **Event** object). It takes the following values: Type – String, range – available parameters for the objects of the specified type.
2. **value** - Required argument. The value to be assigned to the parameter specified in the param argument. It takes the following values: Type – String, range – depends on the parameter.

Usage examples

See the example for the [GetParam method](#).

11.2.7 The MsgToString method

The MsgToString method transforms **MsgObject** objects (including the static **Event** object) into a String variable.

Method call syntax

```
function MsgToString() : String
```

Method arguments: no arguments.

Usage examples

Problem. Send the messages about all events registered for Microphone 1, to a specified e-mail address.

Note

The **Short Messages Service** is supposed to be installed and working properly.

```
if (Event.SourceType == "OLXA_LINE" && Event.SourceId == "1")
{
    var msgstr = Event.MsgToString();
    DoReactStr("MAIL_MESSAGE", "1", "SEND", "subject<Microphone 1>,body<" + msgstr + ">");
    DoReactStr("MAIL_MESSAGE", "1", "SEND", "");
}
```

11.2.8 The StringToMsg method

The StringToMsg method transforms a String variable into an **MsgObject** object.

Method call syntax

```
StringToMsg(msg : String) : MsgObject
```

Method arguments:

msg - Required argument. A String type variable to be transformed into an **MsgObject** object. It takes the following values: Type – String; range – character string that matches the syntax for **MsgObject** representation:

"objtype|id|action|param1<value1>,param2<value2>...", where

objtype – object type;

id – object identification number;

action – event or action for the object;

param1<value1>,param2<value2> - list of parameters and their values. Elements of the list are separated by commas with no white space. If no parameters need to be specified, an empty string is used after the vertical line (|), for example:

"CAM|1|MD_START|"

Usage examples

Problem. Upon an alarm in Sensor 1 or 3, start recording audio from Microphone 1. Upon an alarm in Sensor 2 or 4, start recording audio from Microphone 2.

```

if (Event.SourceType == "GRAY" && Event.Action == "ALARM")
{
    var audioid;
    if (Event.SourceId == "1" || Event.SourceId == "3")
    {
        audioid = "1";
    }
    if (Event.SourceId == "2" || Event.SourceId == "4")
    {
        audioid = "2";
    }
    var str = "OLXA_LINE|" + audioid + "|ARM|";
    var msg = CreateMsg();
    msg.StringToMsg(str);
    NotifyEvent(msg);
}

```

11.2.9 The StringToParams method

The `StringToParams` method transforms a `String` variable into the list of parameters and overwrites the existing parameter list of the **MsgObject** object.

Method call syntax

```
StringToParams(String params)
```

Method arguments:

params - Required argument. A `String` type variable to be transformed into a list of parameters for the **MsgObject** object. It takes the following values: `String` variables matching the syntax for **MsgObject** parameter list representation:

"param1<value1>,param2<value2>...", where

param1<value1>,param2<value2> - list of parameters and their values. Elements of the list are separated by commas with no white space. If no parameters need to be specified, an empty string is used after the vertical line (|), for example:

"CAM|1|MD_START|"

Usage examples

Example. Upon registration the connection ("Attach") event for any camera, in the system retrigger the "Attach" event with modified **Number of the PTZ device** (`telemetry_id`) and **Number of the microphone for synchronous recording** (`audio_id`) parameters. The values should be equal to the corresponding camera numbers plus 1.

```

if (Event.SourceType == "CAM" && Event.Action == "ATTACH")

```

```

{
var i;
for (i=1,i<=4;i=i+1)
{
var msg = Event.Clone();
var str = "telemetry_id<" + (i+1) + ">,audio_id<" + (i+1) + ">";
msg.StringToParams(str);
NotifyEvent(msg);
}
}

```

11.2.10 The Clone method

The Clone method creates a copy of an **MsgObject** or **Event** object.

Method call syntax

```
Clone() : MsgObject
```

Method arguments: no arguments.

Usage examples

Problem. When Relay №1 closes, start video recording from Camera 1 and close Relay №2. When Relay №1 opens, start video recording from Camera 2 and open Relay №2.

```

if (Event.SourceType == "GRELE" && Event.SourceId == "1")
{
var msgevent = Event.Clone();
if(Event.Action == "ON")
{
msgevent.SourceId = "2";
DoReact(msgevent);
DoReactStr("CAM","1","REC","");
DoReactStr("GRELE","2","ON","");
}
if(Event.Action == "OFF")
{
msgevent.SourceId = "2";
DoReact(msgevent);
DoReactStr("CAM","2","REC","");
DoReactStr("GRELE","2","OFF","");
}
}
}

```

11.2.11 The GetObjectIds method

GetObjectIds method is responsible for receiving identifiers from all the objects of a specified type.

Method call syntax:

```
function GetObjectIds(objectType : String)
```

A line is replied :

```
CAM||COUNT|id.3<5>,id.count<4>,id.0<2>,id.1<3>,id.2<4>
```

where **id.count<>** – number of ID objects,

id.[count]<> – ID object.

Method's arguments:

objectType –required argument. Set the type of the system object, for which the value of the given parameter should be given back ("CAM","GRAY","GRABBER" e.t.c.).Accepted values: type String, range is restricted by object types registered in the system.

Example. All the cameras should be armed upon the start of Macros№1.

```
if (Event.SourceType == "MACRO" && Event.SourceId && Event.Action == "RUN")
{
var msg = CreateMsg();
msg.StringToMsg(GetObjectIds("CAM"));
var objCount = msg.GetParam("id.count");
var i;
for(i = 0; i < objCount; i++)
{
DoReactStr("CAM", msg.GetParam("id." + i), "ARM", "");
}
}
```

11.2.12 The GetObjectParams method

GetObjectParams method is designed for getting the object's parameters.

Method call syntax:

```
function GetObjectParams(objectType : String, objectId : String)
```

Method arguments:

1. **objectType** – required argument. Set the type of the system object ("CAM", "GRAY", "GRABBER" e.t.c.), for which the type of a parent object should be given back. Accepted values: type String, range is restricted by object types registered in the system.
2. **objectId** – object's identifier. Accepted values: String type.

Example. It is necessary to check the color control of camera №2 upon the start of Macros№1. If camera 2 is a color one, set it to recording.

```
if (Event.SourceType == "MACRO" && Event.SourceId && Event.Action == "RUN")
{
var msg = CreateMsg();
msg.StringToMsg(GetObjectParams("CAM", "2"));
if(msg.GetParam("color") == "1")
{
DoReactStr("CAM", "2", "REC", "");
}
}
```

11.2.13 The SourceType property

The SourceType property stores the system type of the **MsgObject** or **Event** object.

Property call syntax

```
SourceType : String
```

Usage examples

Problem. When Relay 1 closes (for example, the button connected to the relay is pressed), print the frames from Cameras 1 and 2.

```
if (Event.SourceType == "GRELE" && Event.SourceId == "1" && Event.Action == "ON")
{
  //activating the Camera 1 window
  DoReactStr("MONITOR","1","ACTIVATE_CAM", "cam<1>");
  //printing the frame from Camera 1
  DoReactStr("MONITOR","1","KEY_PRESSED","key<PRINT>");
  //activating the Camera 2 window
  DoReactStr("MONITOR","1","ACTIVATE_CAM", "cam<2>");
  //printing the frame from Camera 2
  DoReactStr("MONITOR","1","KEY_PRESSED","key<PRINT>");
}
```

11.2.14 The SourceId property

The SourceType property stores the identification number of the **MsgObject** or **Event** object.

Property call syntax

```
SourceId : String
```

Usage examples

See the example for the [SourceType](#) property.

11.2.15 The Action property

The Action property stores the action or event of the **MsgObject** or **Event** object.

Property call syntax

```
SourceId : String
```

Usage examples

See the example for the [SourceType](#) property.

12 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. [The Script object. Programming using the JScript language](#).

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.