Intellect integration guide (HTTP API, IIDK, ActiveX, HTTP Server, Axxon Next)

Last update 01/07/2023

Table of contents

1 Intellect integration guide. Introduction

2 Hardware and Software Module Integration

3 CamMonitor.ocx ActiveX Control

4 Intellect HTTP API

5 Intellect HTTP-Server

6 Axxon Next integration

7 Configuring RabbitMQ

8 Intellect software Integration Guide. Postscript

9 APPENDIX 1. DDI file structure

10 APPENDIX 2. NissObjectDLLExt and CoreInterface class declarations

Download DEMO module

Note

This module is used as an example of working with IIDK—see Hardware and Software Module Integration.

11 Intellect integration guide. Introduction

This guide covers the information on how to provide interaction of *Intellect* with external systems. *Intellect* has the following solutions:

1. **IIDK Interface**. It is used to integrate functional modules into the system. Functional modules perform the following tasks:
    a. Adding new security hardware to the system.
    b. Implementing new service functions (security hardware management).

   The module integration steps are examined using a demonstration module *DEMO* as an example. Its source files are attached to the documentation.
   You can download DEMO module at Intellect integration guide (HTTP API, IIDK, ActiveX, HTTP Server, Axxon Next).
2. **CamMonitor.ocx ActiveX Control**. The component is similar in every way to the **Video surveillance monitor** interface object. It allows you to manage cameras, view the archive, etc.
3. **HTTP API**. The software interface allows you to send commands and receive data from *Intellect* using the HTTP requests.
4. **RabbitMQ**. It allows you to send and receive messages in *Intellect* using the RabbitMQ message broker.
5. ***Axxon Next* integration**. It allows you to receive data from *Axxon Next* video cameras.

12 Hardware and Software Module Integration

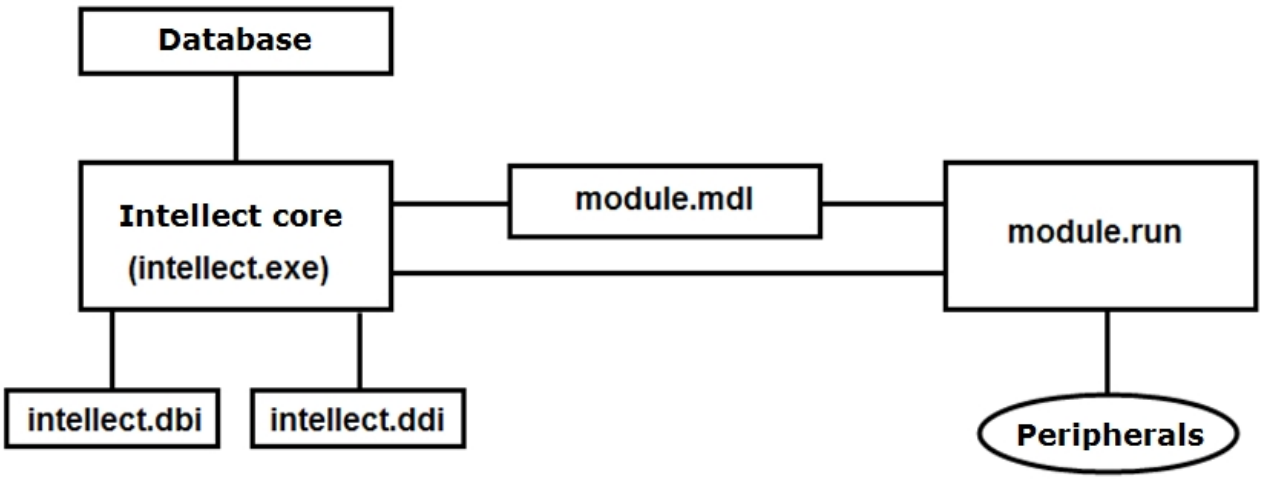
12.1 Integrating hardware and software modules with Intellect

12.1.1 General information on hardware and software modules integrating

To integrate a hardware and software (functional) module into Intellect, perform the following steps:

1. Edit the DBI file.
2. Edit the DDI file.
3. Prepare a module.mdl file, where “module” is the name of the module to be integrated (this file is a transformed DLL file).
4. Prepare an executable file, module.run, where “module” is the name of the module to be integrated (this file is a transformed EXE file).
5. Copy module.mdl and module.run to the *Intellect\Modules* folder.

Figure below shows a diagram of interaction between a functional module and the system core.



The DBI and DDI files contain information on the integrated functional modules (objects); this information is needed for the operation of the system core. The DBI file describes the structure of Intellect's configuration database. The DDI file describes the objects and their parameters. When an object is integrated, the name of the object, its parameters, and the related system events and reactions are added to these files.

The MDL file is used for working with one type of objects: it allows you to create, delete, and modify object parameters (during setup or operation), save them in the database, and perform several special operations. The MDL file also ensures that the parameters of created/modified objects are sent to the executable file (the RUN file), and contains the configurations of the object setup panels.

The executable RUN file interacts with devices, passes event information to the core, and enables device management.

In this document, we describe the steps for module integration by using the DEMO module, which emulates working with virtual hardware. This module includes devices with unique addresses for accessing and polling these devices. The system allows you to perform a number of actions with devices and to pass all their events to the system core.

12.1.2 Editing the DBI file

The intellect.dbi file contains the master list of the tables and fields of the database. We recommend that you create your own database template in a separate file and name it intellect.xxx.dbi, where xxx is a unique sequence in the filename. By using a separate file, you avoid double inclusion of the tables and fields after an update to the Intellect software package. On startup of the software package, the DBI files are merged.

12.1.2.1 Adding Objects to Intellect.dbi

Objects are added to intellect.dbi as follows:

1. Go to *Intellect's* root folder and open the intellect.dbi file with a text editor.
2. Add the objects to intellect.dbi. For each object, you must supply its name (used for identification) in brackets and then declare its fields. Below is the field declaration syntax:
   <Field name>, <Type> [, <Size>]

   > **Note.**
   >
   > The **Size** may be set for fields of the CHAR type only.

   The table below shows the fields mandatory for all objects in *Intellect.*

Field	Description
id	Unique object ID
name	Object name
parent_id	Parent object ID
flags	Parameter for internal system use

   > **Attention!**
   >
   > The flags field may not be used by external applications.

   The following table describes the allowed data types.

Data type	**Description**
BIT	Used for creating a flag field that takes a logical value, Yes or No.
CHAR	Used for fields that contain short character sequences.
DATETIME	Used for fields that contain dates and times. The date format is YYYY-MM-DD and the time format is HH:MM:SS.XXX.
DOUBLE	Used for fields that contain floating-point numbers.
INTEGER	Used for fields that contain integer numbers.
TEXT	Used for fields that contain text strings.

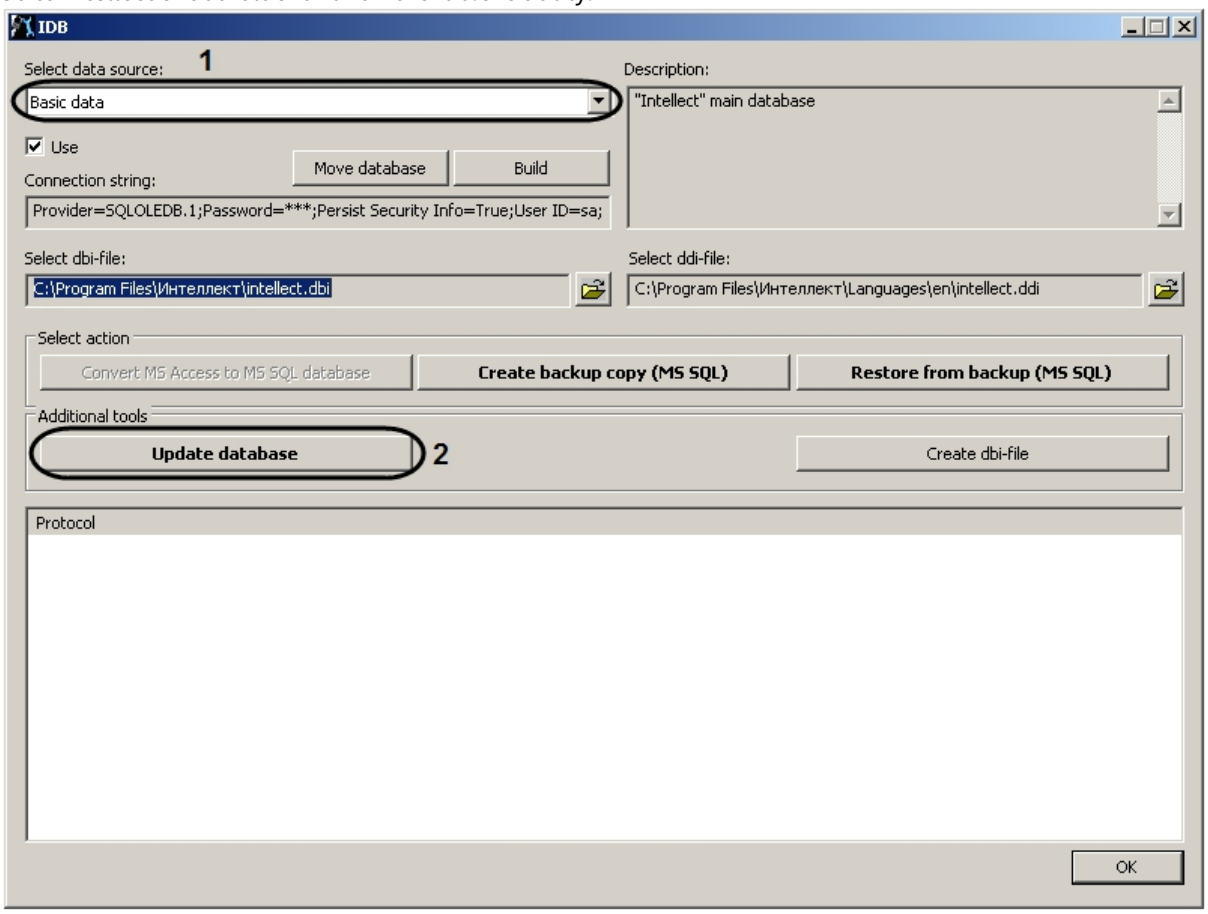
   Beside the mandatory fields, the objects of the DEMO module contain the following fields:
   a. **port** – COM port address;
   b. **address** – device address.

Figure below shows sample object additions and field declarations in intellect.dbi.

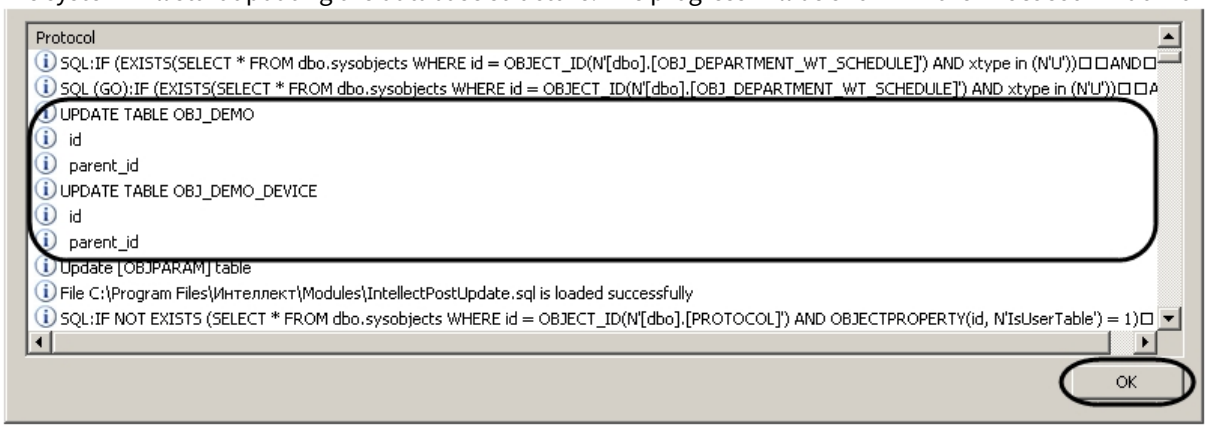
```
[OBJ_DEMO]
id, CHAR, 16
name, CHAR, 60
parent_id, CHAR, 16
flags, INTEGER
port, CHAR, 5

[OBJ_DEMO_DEVICE]
id, CHAR, 16
name, CHAR, 60
parent_id, CHAR, 16
flags, INTEGER
address, INTEGER
```

3. Save the changes to the intellect.dbi file.
4. Go to Intellect's root folder and run the idb.exe utility.



5. In the **Select data source** list, select **Basic data** (1).
6. Click the **Update database** button (2).
   The system will start updating the database structure. The progress will be shown in the **Protocol** window of idb.exe.



7. Click **OK** to close idb.exe.

As a result of the database structure update, tables are created in *Intellect's* configuration database.

12.1.2.2 Using the ddi.exe Tool to Work with DBI files

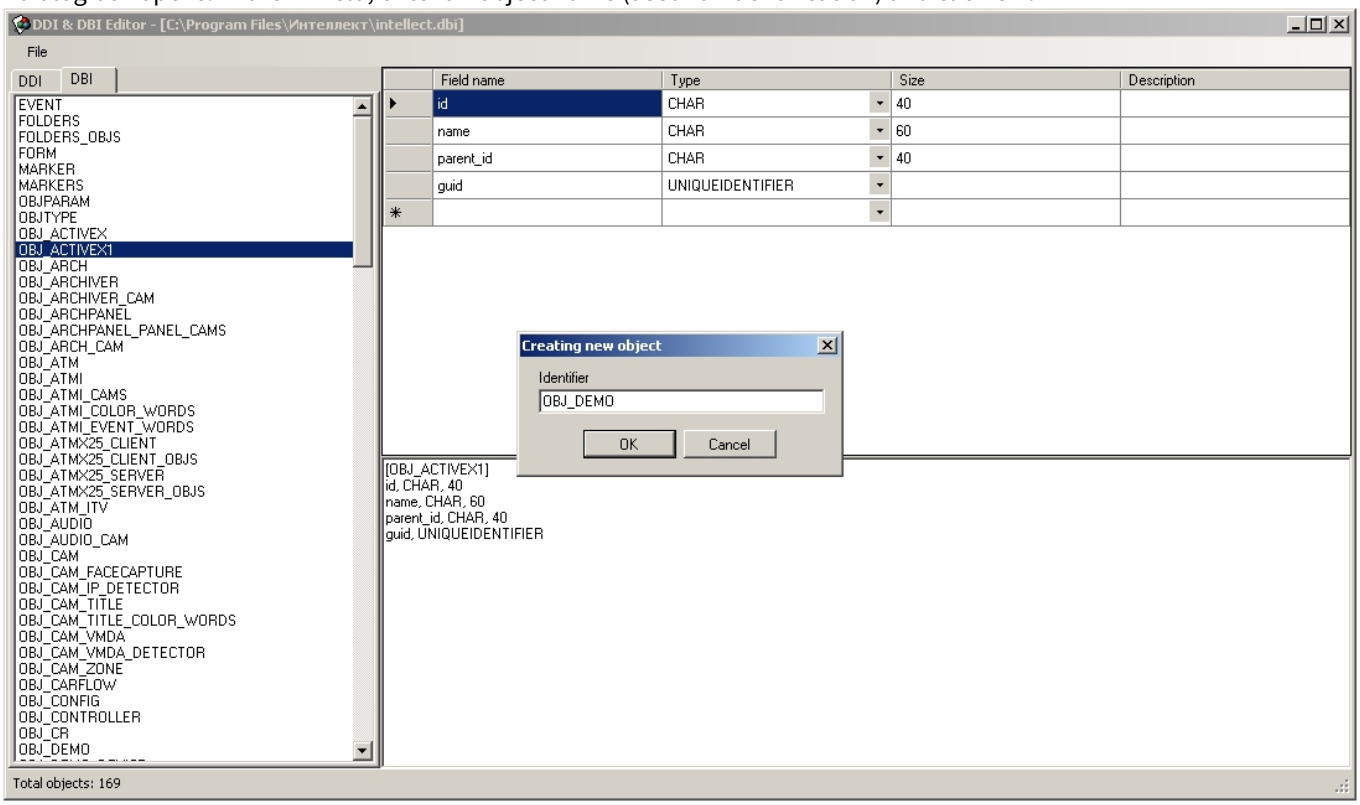
To add an object to the DBI file by using the *ddi.exe* utility, do the following:

1. Go to the *Intellect\Tools* folder and run *ddi.exe*.
2. In the program window, select the **DBI** tab.
3. In the **File** menu, select **Open**. The **Open** dialog box appears.
4. Go to Intellect's root folder and select the intellect.dbi file. The *ddi.exe* window shows a list of objects.

5. To add the new object, in the list's context menu, select **Add**.

> **Note:**
>
> You may add a new object by pressing the **Insert** key as well.

6. A dialog box opens. In the **ID** field, enter an object name (used for identification) and click **OK.**



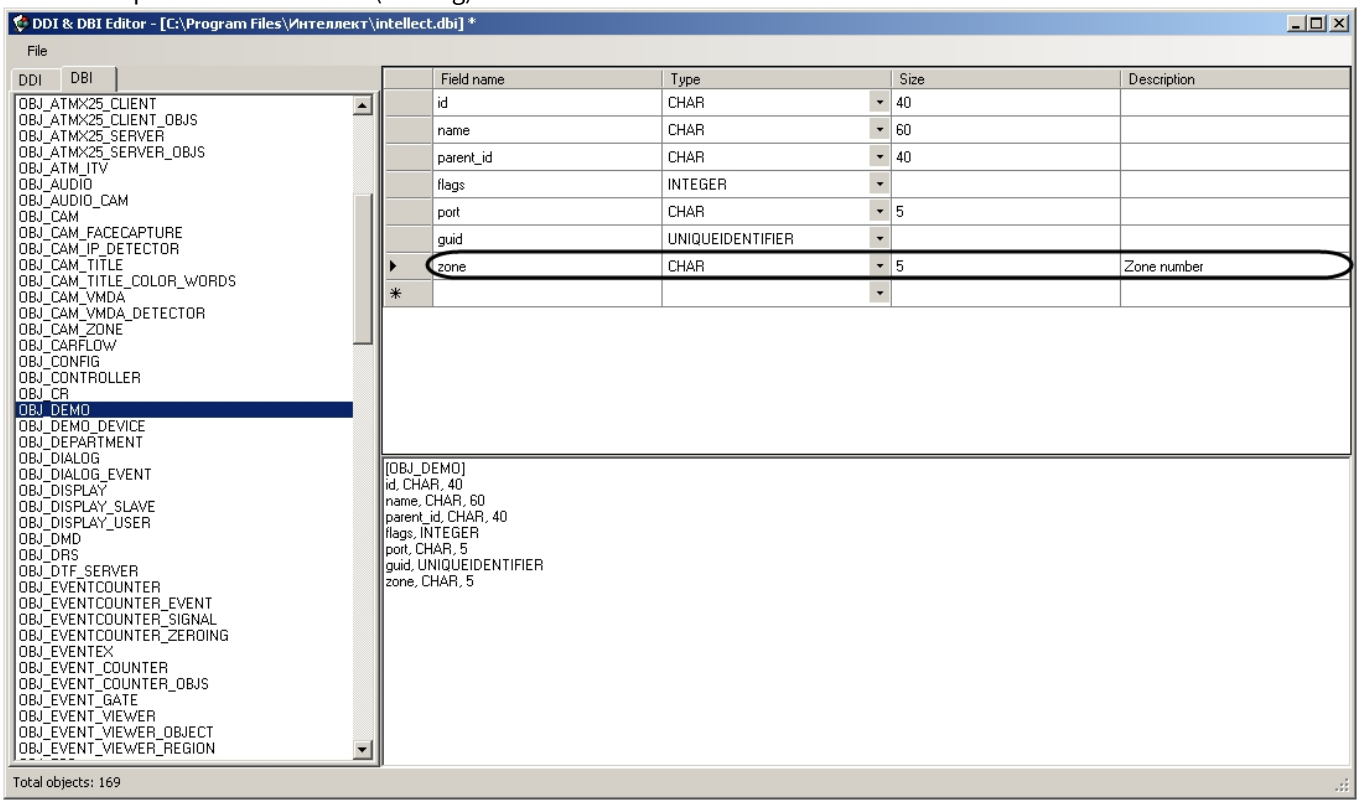
> **Note:**
>
> The mandatory fields are automatically added to the created object (see Section Adding Objects to Intellect.dbi).

The object has now been added to the DBI file.

To add a field:

1. In the left part of the *ddi.exe* window, select an object.

2. Add a description of the new field (a string) to the table.



3. To save the changes, in the **File** menu, select **Save.**

The new field is now added.

> **Attention!**
>
> After making changes to the DBI file, you must update the database structure by using idb.exe (see Adding Objects to Intellect.dbi).

12.1.3 Editing the DDI file

The DDI file is an XML file that contains the following object information:

1. Reactions (that is, actions that the objects may perform).
2. Events that the objects may generate.
3. States of the objects.
4. Event-driven rules for state transition.
5. The names of the BMP files that are used for visualizing the objects on the *Map*.

The intellect.ddi file contains the properties of Intellect's main objects. For your own objects, we recommend creating a separate file, intellect.xxx.ddi, where xxx is a unique part of the filename. By using a separate file, you avoid double inclusion of the properties after an update of the Intellect software package. On startup of the software package, the DDI files are merged.

If an object is duplicated in several ddi-files, then at Intellect software startup the properties of the object from the last file are applied in accordance with the sorting of files by name. For example, if an object is described in files intellect.xxx.ddi, intellect.xxx1.ddi and intellect.xxx2.ddi, the properties from the intellect.xxx2.ddi will be applied.

12.1.3.1 Adding Object Information to intellect.ddi

This section shows how to add information on the **DEMO** object to intellect.ddi by using a text editor.

To add information on the **DEMO** object, do the following:

1. Go to *Intellect**Languages*\en folder and open the intellect.dbi file with a text editor.

2. Into the **<DataSetDDI>** section, add a child element, **<Objects>**, which contains an object description.

```
<Objects>
  <ObjectName>DEMO</ObjectName>
  <VisibleName>Demo object</VisibleName>
  <GroupName></GroupName>

  <Events>
    <EventName>LOST</EventName>
    <EventDescription>Connection lost</EventDescription>
    <IsSoundEnabled>false</IsSoundEnabled>
    <IsNetworkDisabled>false</IsNetworkDisabled>
    <IsProtocolDisabled>false</IsProtocolDisabled>
    <IsWindowsLogEnabled>false</IsWindowsLogEnabled>
  </Events>
  <Events>
    <EventName>RESTORE</EventName>
    <EventDescription>Connection restored</EventDescription>
    <IsSoundEnabled>false</IsSoundEnabled>
    <IsNetworkDisabled>false</IsNetworkDisabled>
    <IsProtocolDisabled>false</IsProtocolDisabled>
    <IsWindowsLogEnabled>false</IsWindowsLogEnabled>
  </Events>
  <Icons>
    <FileName>demo</FileName>
    <IconName>demo</IconName>
  </Icons>
  <States>
    <StateName>DETACHED</StateName>
    <ImgName>detached</ImgName>
    <StateDescription>Armed</StateDescription>
    <IsStateFlashing>false</IsStateFlashing>
  </States>
  <States>
    <StateName>NORMAL</StateName>
    <ImgName>normal</ImgName>
    <StateDescription>Disarmed</StateDescription>
    <IsStateFlashing>false</IsStateFlashing>
  </States>
  <Rules>
    <EventName>RESTORE</EventName>
    <FromStateName>DETACHED</FromStateName>
    <ToStateName>NORMAL</ToStateName>
  </Rules>
  <Rules>
    <EventName>LOST</EventName>
    <FromStateName>NORMAL</FromStateName>
    <ToStateName>DETACHED</ToStateName>
  </Rules>
</Objects>
```

> **Note.**
>
> For the **DEMO** object, the <**Reacts**> sections is missing, because this object does not perform any actions.

> **Note.**
>
> The DDI file elements are described in detail in APPENDIX 1. DDI file structure.

3. Save the changes to the intellect.ddi file.

The information on the **DEMO** object has now been added to intellect.ddi.

> **Attention!**

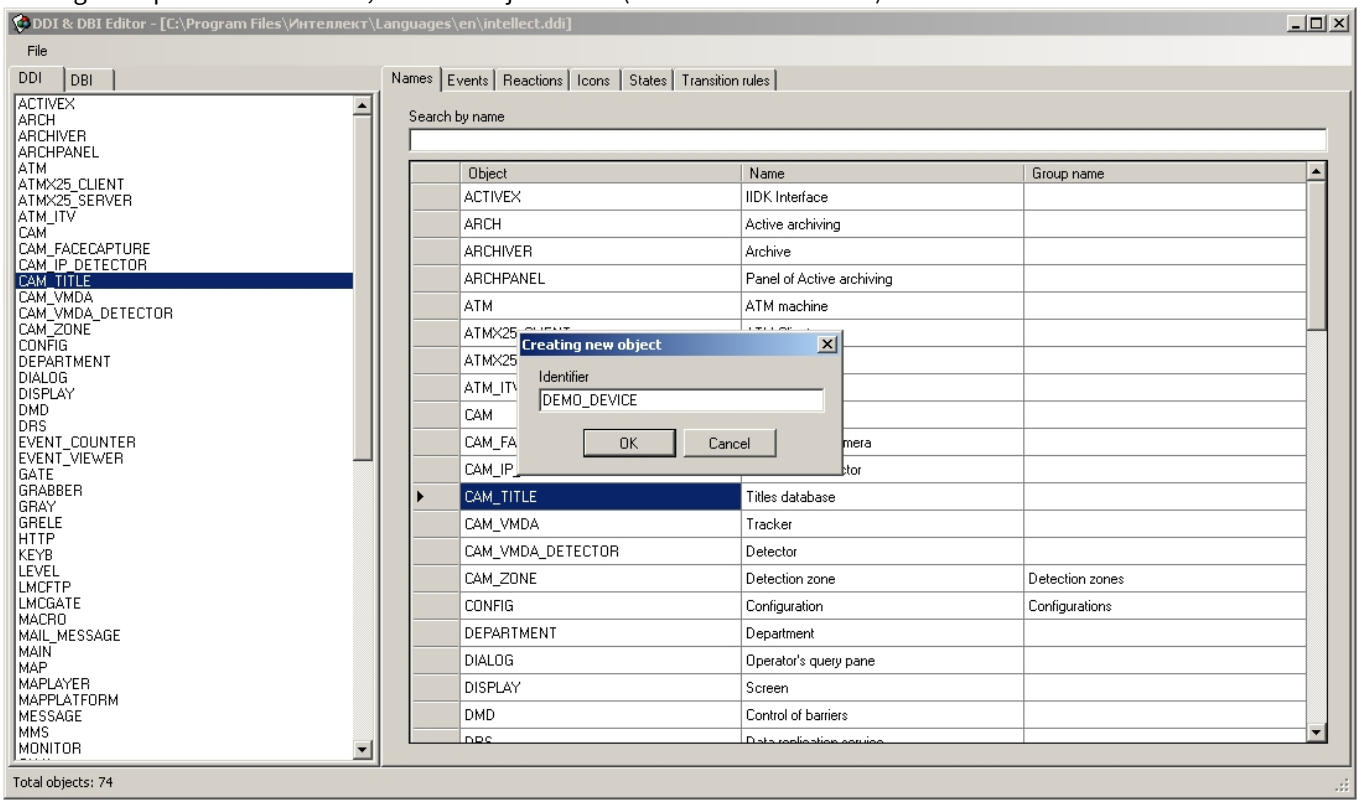
After making changes to the DDI file, you must update the database structure by using idb.exe (see Steps 4–7 in Section Adding Objects to Intellect.dbi).

12.1.3.2 Using the ddi.exe Tool to Work with DDI files

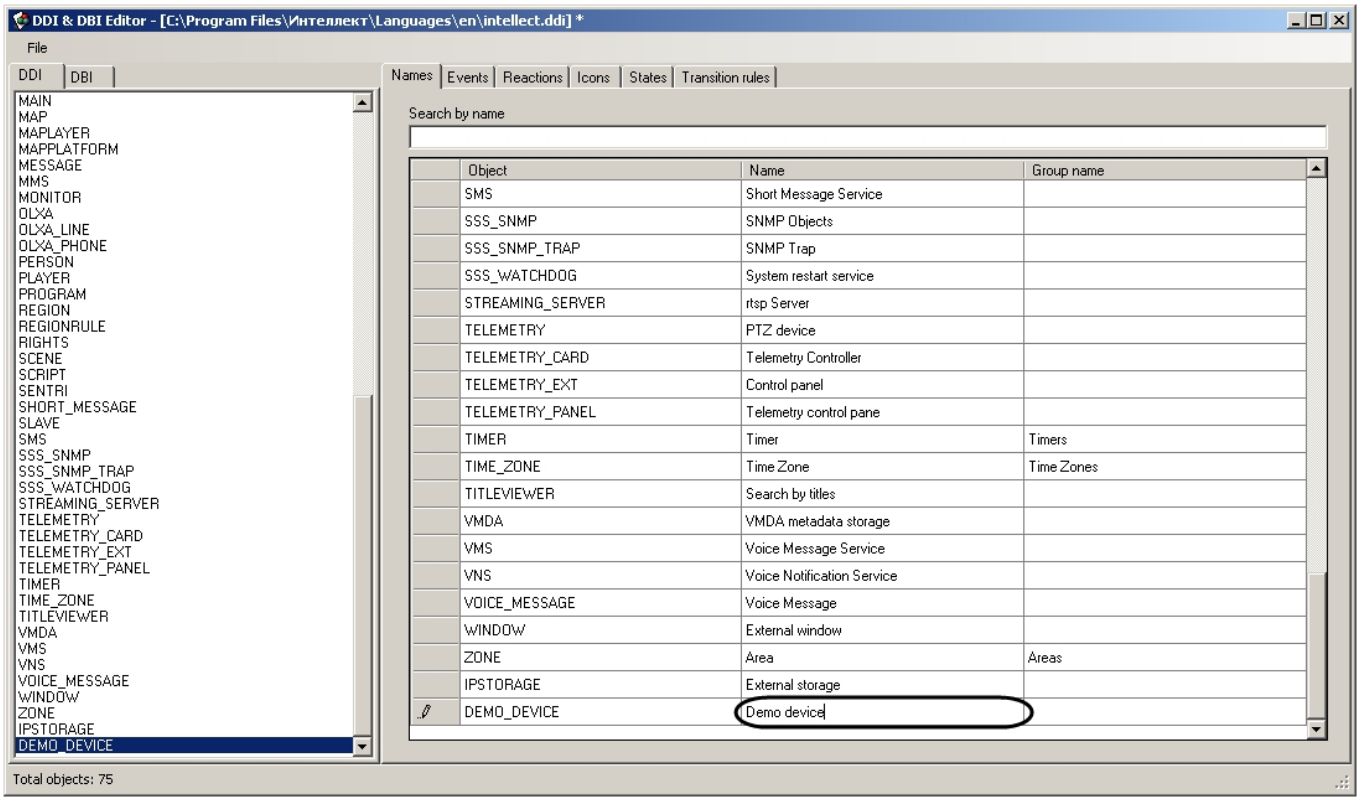
This section shows how to add information on the **DEMO_SERVICE** object to intellect.ddi by using the *ddi.exe* tool.

To add information on the **DEMO_DEVICE** object, do the following:

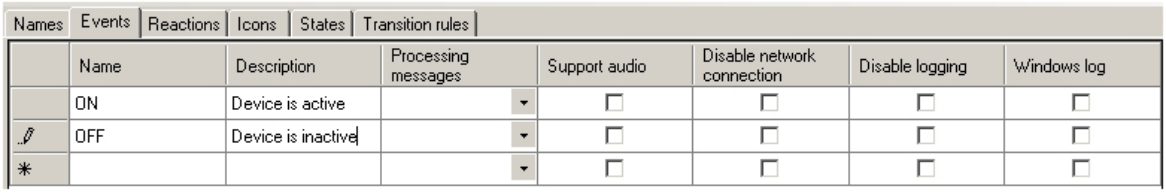
1. Open intellect.ddi file in one of the following ways:
   a. Go to the *Intellect\Tools* folder and run *ddi.exe*.
      i. In the program window, select the **DDI** tab.
      ii. In the **File** menu, select **Open**. The **Open** dialog box appears.
      iii. Go to the *Intellect\Languages*\en folder and select intellect.ddi. The window of *ddi.exe* shows a list of objects.
   b. Open the *Intellect\Tools* folder in Windows Explorer or any other file manager, then double-click the intellect.ddi file.
2. Add the object by selecting **Add** in the list's context menu or by pressing the **Insert** key.
3. A dialog box opens. In the **ID** field, enter an object name (used for identification) and click **OK**.



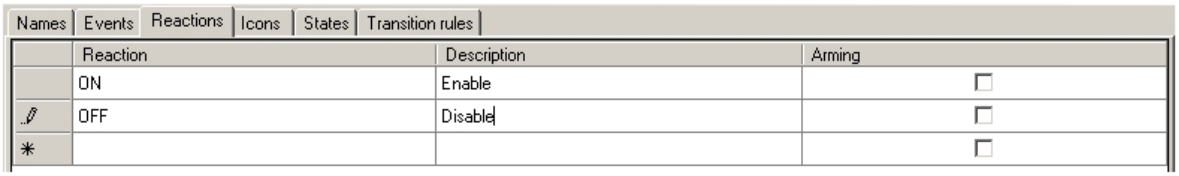
   The DEMO_DEVICE object is now shown in the list of objects.
4. In the **Names** tab, enter an object name.



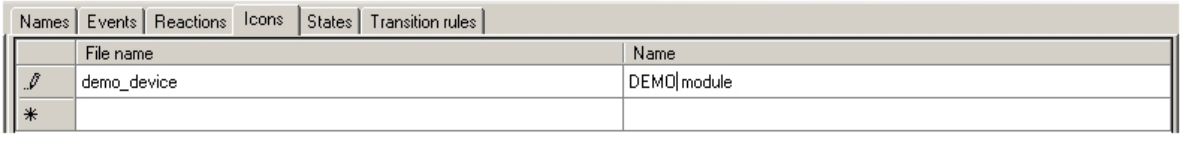
5. In the relevant tabs, add information on the DEMO_DEVICE object.
   a. In the **Events** tab, add the **ON** and **OFF** events.



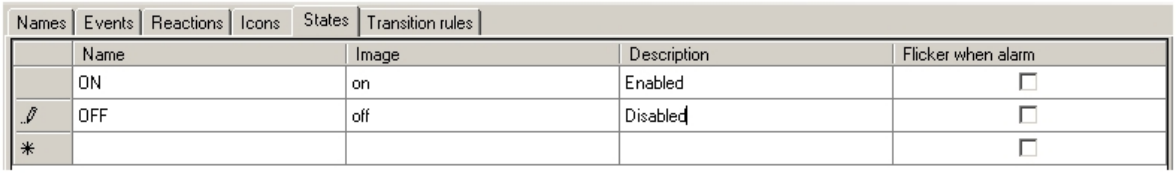
   b. In the **Reactions** tab, add the **ON** and **OFF** actions.



   c. In the **Icons** tab, enter a BMP file name (the part that serves as an image ID). Image IDs allow you to use multiple BMP files to show objects of the same type on the **Map.**



   d. In the **States** tab, add the **ON** and **OFF** states. To show an object state on the **Map,** enter a BMP file name (the part that serves as an ID of the state).



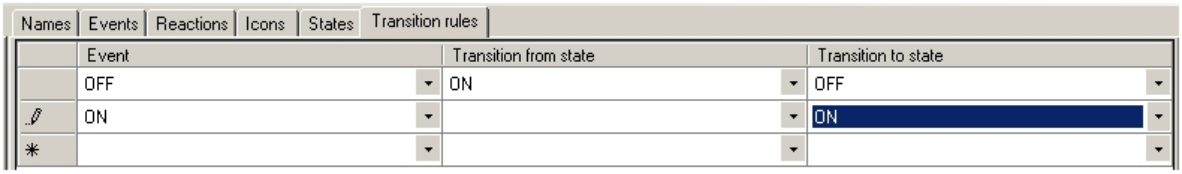
Note.

The names of the BMP files in the Intellect\Bmp folder must have the following format:
<Image ID>_<State ID>
If an image ID is not set, the BMP file name must be the following:
<Object ID>_<State ID>

Note.

The Map may show objects using lines (that is, without using BMP files). In this case, when an object changes its state, the line color changes. For a state, the color (RGB) is set as follows:
<State>$R:G:B

e. In the **Transition Rules** tab, set a rule for transitioning from one state to another after a certain event.



Note.

If the **Transition from state** field is left blank, the rule will apply to all starting states.

6. To save changes, in the **File** menu, select **Save.**

The information on the DEMO_DEVICE object is now added.

Note:

The fields of the ddi.exe tables are described in detail in APPENDIX 1. DDI file structure.

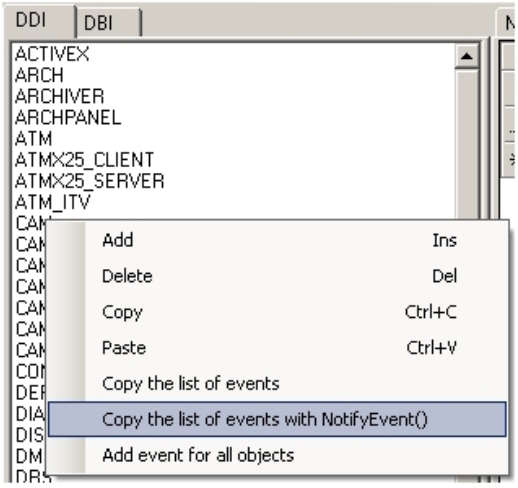
Attention!

After making changes to the DDI file, you must update the database structure by using idb.exe (see Steps 4–7 in Section Adding Objects to Intellect.dbi).

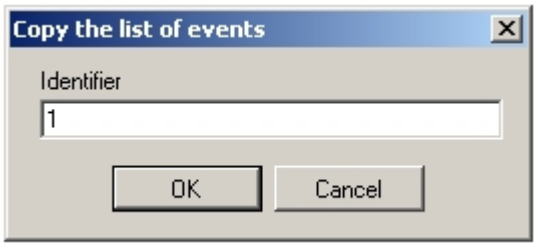
12.1.4 Additional Functionality of the ddi.exe Utility

The *ddi.exe* tool allows you to conveniently delete, add, and edit object properties (such as events and reactions), and copy them to the clipboard. In addition, you can copy object events to the clipboard, as a parameter of the *NotifyEvent* function. To do this, follow the steps below:

1. In the object list's context menu, select **Copy the list of events with NotifyEvent()**.



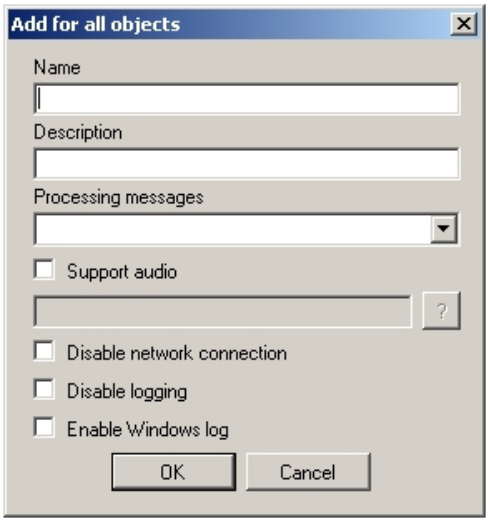
2. A dialog box opens. In the dialog box, enter the object ID to be used by the NotifyEvent function and click **OK.**



The event list is now copied. The clipboard contains object events in the format shown in the figure below.

```
NotifyEvent("CAM","1","ARM");
NotifyEvent("CAM","1","ATTACH");
NotifyEvent("CAM","1","BLINDING");
NotifyEvent("CAM","1","DETACH");
NotifyEvent("CAM","1","DISARM");
NotifyEvent("CAM","1","DISC_MOUNT");
NotifyEvent("CAM","1","DISC_UNMOUNT");
NotifyEvent("CAM","1","FILE_REC_ERROR");
NotifyEvent("CAM","1","MD_START");
NotifyEvent("CAM","1","MD_STOP");
NotifyEvent("CAM","1","PRINT");
NotifyEvent("CAM","1","REC");
NotifyEvent("CAM","1","REC_STOP");
NotifyEvent("CAM","1","RECORDER_OFF");
NotifyEvent("CAM","1","RECORDER_ON");
NotifyEvent("CAM","1","UNBLINDING");
```

To add an event for all objects, in the context menu, select **Add Event for All Objects**. The **Add for All Objects** dialog box opens. In the dialog box, specify the parameters of the new event.



To add objects from other DBI and DDI files, in the **File** menu, select **Insert from File**.

12.1.5 Creating MDL files

To create an MDL file, use two classes:

1. *NissObjectDLLExt*. All objects inherit from this class, whose virtual methods are redefined.
2. *CoreInterface*. The methods of this class are used to get parameters of the system's objects.

The declared classes and methods are contained in the nissdlle.h header file. The code contained in the nissdlle.h file is shown in APPENDIX 2. NissObjectDLLExt and CoreInterface class declarations.

Note:

The methods of a class are the procedures and functions declared in its body.

The methods of the *NissObjectDLLExt* class are described in the table below.

Method	Description	Example
CoreInterface* m_pCore	A pointer to the core interface	
virtual BOOL IsWantAllEvents()	Returns TRUE if the OnEvent function receives events from all objects; returns FALSE if the function receives events from its own object only.	If "CAM,GRABBER" is passed as a parameter, when settings of these objects are modified, the DEMO object receives the following messages: DEMO|1|UPDATE_CAM|parameters of the camera DEMO|1|UPDATE_GRABBER|parameters of the video capture card
virtual CString DescribeSubscribeObjectsList()	The method accepts a comma-separated list of objects. When an object from the list is modified, the current object is notified.	
virtual CString GetObjectType()	Returns the object type	`virtual CString GetObjectType()` `{ return "DEMO"; }`
virtual CString GetParentType()	Returns the parent object type	`virtual CString GetParentType()` `{ return "SLAVE"; }`

virtual int GetPos()	Returns the position of the object in the *intellect.sec* key file.
	
	Attention! This parameter must be set in consultation with AxxonSoft.
		`virtual int GetPos()`
		`{ return -1; }`
		
		Note: If Intellect is run in the demo mode, the function returns ***-1***

virtual CString GetPort()	Returns the number of the port used for communication between the object and the core. ***Attention! This parameter must be set in consultation with AxxonSoft.***	`virtual CString GetPort()` `{ return "1100"; }`

Method	Description	Example
virtual CString GetProcessName()	Returns the process name. Used by the core to search for and automatically run the executable module on startup of the system and initialization of the module	`virtual CString GetProcessName()` `{ return "demo"; }`
virtual CString GetDeviceType()	Determines the type of the object and its behavior. ACD – objects of this type receive all events related to the creation, modification, and deletion of the following objects: **Users**, **Time Zone**, and **Access Levels** ACD2 – a type similar to ACD, providing the additional (provided by the core) functionality of deleting temporary (fixed-term) cards The ACR type means that the object is a reader	All objects of the ACR type are available in the **Access Point** drop-down list
virtual BOOL HasChild()	Returns TRUE if the object has child objects, FALSE otherwise.	`virtual BOOL HasChild()` `{ return TRUE; }`
virtual UINT HasSetupPanel()	Returns TRUE if the object has a setup panel, FALSE otherwise	`virtual UINT HasSetupPanel()` `{ return TRUE; }`
virtual void OnPanelInit(CWnd*)	Used when the object's setup panel is initialized. The parameter is a pointer to the setup panel's window.	

virtual void OnPanelLoad(CWnd*,Msg&)	Used when the setup panel is loaded for setting the parameters of the object. The parameters are the setup panel's window and a message used to pass the parameters and fill in the relevant fields of the setup panel.	`virtual void OnPanelLoad(CWnd* pwnd,Msg& params)`
		`{`
		`CString s;`
		`s = arams.GetParam("port");`

Method	Description	Example

		`pwnd->GetDlgItem(IDC_PORT)-> SetWindowText(s);`
		
		`}`

virtual void OnPanelSave(CWnd*,Msg&)	Used when the setup panel is saved for saving the parameters of the object. The parameters are a pointer to the setup panel's window and a reference to a message used to pass the parameters and save them in a database.	`virtual void OnPanelSave(CWnd* pwnd,Msg& params)`
		
		`{`
		
		`CString s;`
		
		`pwnd-> GetDlgItem(IDC_PORT)-> GetWindowText(s);`
		
		`params.SetParam("port",s);`
		
		`}`

virtual void OnPanelExit(CWnd*)	Used when the object's setup panel is closed ("exited"). The parameter is a pointer to the setup panel's window.	

virtual void OnPanelButtonPressed(CWnd*,UINT)	Used to handle clicks on the setup panel's buttons. The parameters are a pointer to the setup panel's window and a button ID.
	
	*Note: A button ID must be a number equal to or greater than **1151**. For example, the Resource.h file defines the ID of the **Test** button as follows:*
	
	#define IDC_TEST 1151
		`Virtual void OnPanelButtonPressed (CWnd* pwnd,UINT id)`
		
		`{`
		
		`if(id==IDC_TEST)`
		
		`{ React react("DEMO",Id,"TEST");`
		
		`m_pCore->DoReact(react); }`
		
		`}`

	If a button click is to open your own dialog box created in the same MDL file, you must first use the code shown in the example below.	`HINSTANCE prev_hinst = AfxGetResourceHandle();`

Method	Description	Example

		HMODULE hRes = GetModuleHandle("demo.mdl");
		
		If (hRes) AfxSetResourceHandle (hRes);
		
		//Code for showing a dialog box:
		
		CXXXDialog dlg;
		
		dlg.DoModal();
		
		AfxSetResourceHandle(prev_hinst);

virtual BOOL IsRegionObject()	Shows whether the object supports Intellect's regions. Regions are used to group objects. They can also be used in the report system.	
virtual BOOL IsProcessObject()	Shows whether the object supports starting and running multiple executable modules simultaneously. For example, this may be used for starting a separate module for each COM port. *Note: We recommend using one RUN file. This makes it easier to debug and modify the module.*	

virtual void OnCreate(Msg&)	Used when the object is created. The parameter is a reference to a message that contains object information. The method is also used to set default parameters.	virtual void OnCreate (Msg& msg)
		
		{
		
		msg.SetParam ("port","COM1");
		
		}

virtual void OnInit(Msg&)	Used when the object is initialized. The parameter is a reference to a message that contains object information.	virtual void OnInit (Msg& msg)
		
		{
		
		OnChange (msg, msg);
		
		}

Method	Description	Example

virtual void OnChange(Msg&,Msg&)	Used when the object is changed. The first and second parameters are references to messages that contain object information before and after the change, respectively.	`virtual void OnChange(Msg& msg, Msg& prev)`
		`{`
		`React react (msg.GetSourceType(), msg.GetSourceId(),"INIT");`
		`react.SetParam("port" ,msg.GetParam("port"));`
		`m_pCore->DoReact(react);`
		`}`

virtual void OnDelete(Msg&)	Used when the object is deleted. The parameter is a reference to a message that contains object information.	`virtual void OnDelete (Msg& msg)`
		`{`
		`React react (msg.GetSourceType(), msg.GetSourceId(),"EXIT");`
		`m_pCore-> DoReact(react);`
		`}`

virtual void OnEnable(Msg&)	Used to handle clicks on the **Disable** button of Intellect's panel when the object is enabled. The parameter is a reference to a message that contains object information.	
virtual void OnDisable(Msg&)	Used to handle clicks on the **Disable** button of Intellect's panel when the object is disabled. The parameter is a reference to a message that contains object information.	

virtual BOOL OnEvent(Event&)	Used to handle the events that are passed as the parameter.	`virtual BOOL`
		`OnEvent(Event& event)`
		`{`
		`If`

Method	Description	Example

		(event.GetAction() == "ACCESS_IN"
		||
		event.GetAction() == "ACCESS_OUT")
		
		{
		Msg per = m_pCore->
		FindPersonInfoByCard(event.GetPara
		m("facility_code"),
		event.GetParam("card"));
		event.SetParam
		("param0", !
		per.GetSourceId().IsEmpty() ?
		per.GetParam("name") :
		event.GetParam("facility_code") +
		event.GetParam("card"));
		event.SetParam("param1",
		per.GetSourceId());
		}
		
		Else
		
		If (event.GetAction() ==
		"NOACCESS_CARD")
		{
		event.SetParam
		("param0",event.GetParam(
		"facility_code") +
		event.GetParam("card"));
		}
		return TRUE;
		}

virtual BOOL OnReact(React&)	Used to handle the reactions that are passed as the parameter.	

The *CreateNissObject(CoreInterface* core)* global function creates instances of the described objects, places them in an array (an instance of *CNissObjectDLLExtArray),* and returns a pointer to this array. This function is used to receive a pointer to the core interface. This pointer is later used by objects to call interface methods:

```
CNissObjectDLLExtArray* APIENTRY CreateNissObject(CoreInterface* core)
{
    CNissObjectDLLExtArray* ar = new CNissObjectDLLExtArray;
    ar->Add(new NissObjectDemo(core));
    ar->Add(new NissObjectDemoDevice(core));
    return ar;
}
```

After loading a DDL file, the core calls the CreateNissObject function and receives pointers to all the objects in use.

All object setup panels are stored in resources as dialogs. Each dialog ID has the format **IDD_object_SETUP**, where **object** is the name of the corresponding object. For example, the ID of the **DEMO object** is **IDD_DEMO_SETUP**, and the ID of the **DEMO_DEVICE** object is **IDD_DEMO_DEVICE _SETUP.**

Note:

If you want for the settings tree to show a special icon for a particular object, in the resources of the DLL file, create a 14x14 **BITMAP** that contains the object name,.

12.1.6 Creating RUN files

Devices are managed by exchanging messages (commands) between RUN files and the system core. For implementing this interaction between software modules and the core, use the *Intellect Integration Developer Kit (IIDK),* which is covered in detail in Section Intellect Integration Developer Kit (IIDK). Other information is provided by the source files of the demonstration module; the files may be found in an appendix to this documentation.

Below is an example of how the *IIDK* is used in the *DEMO* module.

```
CString port = "1100";

CString ip = "127.0.0.1";

CString id = "";

BOOL IsConnect = Connect (ip, port, id, myfunc);

if (!IsConnect)

{

// connection failed

AfxMessageBox("Error");

Return;

}

SendMsg(id,"CAM|1|REC"); // turn on recording for camera 1

SendMsg(id, "DEMO|1|RESTORE"); // restore the connection with the DEMO object
```

```
//turn on the DEMO_DEVICE with address 1

SendMsg(id,"DEMO_DEVICE|1|ON|params<1>,param0_name<address>,param0_val<1>");

Disconnect(id);
```

Attention!

If an MDL file exists, connecting to Intellect's core does not require creating an IIDK Interface object in the system. The connection ID passed is an empty string (in other words, the ID is "").

When a module is unloaded, it receives the **WM_EXIT** event:

#define WM_EXIT (WM_USER+2000)

Use a WinAPI function, *PostThreadMessage,* to catch this message and ensure that the module is unloaded properly. In *VC++* and *MFC,* the **WM_EXIT** event is caught in a subclass of *CWinApp*; in *Delphi* and *CBuilder,* it is caught in a subclass of *TApplication*.

12.1.7 Creating and configuring integrated objects (modules) in Intellect

To create and configure the integrated objects (modules) in *Intellect*, do the following:

1. Place the MDL and RUN files into the *Intellect\Modules* directory.
2. Start *Intellect*.
3. On the basis of the **Computer** object, create the object that was added using the software module. As a result, the object settings panel will be available.

   Note

   You can ask AxxonSoft technical support for a sample integration project with demo objects.
4. Make the necessary changes in the object settings panel.
5. Click the **Apply** button.
6. Repeat steps 3-5 for all required objects.

New objects are now created and configured in *Intellect*.

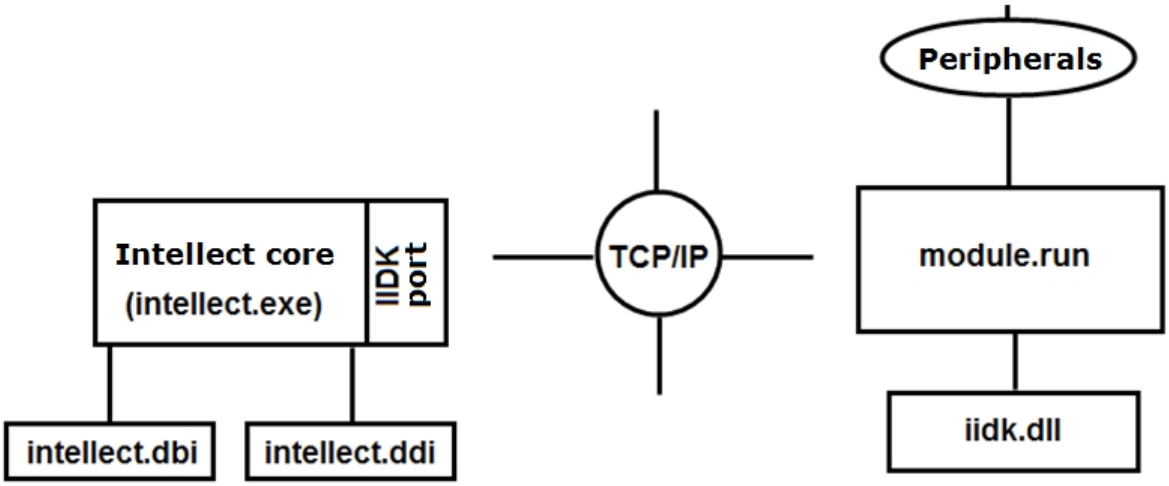
12.2 Intellect Integration Developer Kit (IIDK)

12.2.1 General Information on IIDK

12.2.1.1 Purpose of the IIDK

System expandability is supported by Intellect's software architecture. Expandability allows communication between the core and functional modules (third party information systems) via TCP/IP.

The figure below shows a diagram of interaction between Intellect's core and external software (a functional module).



Interaction is done by exchanging messages in a communication environment; message exchange is implemented by using the *IIDK*.

The *Intellect Integration Developer Kit* (*IIDK*) is a set of development tools for integrating third-party security software into Intellect. This kit allows you to expand the system rapidly and effectively by adding functional modules that support new hardware and new functions.

12.2.1.2 Developer Requirements

To use the *IIDK*, you must:

1. know how to program in C/C++;
2. know the basics of Win32 programming;
3. have an IDE with DLL support (such as *Microsoft Visual C++*, *C++ Builder,* or *DELPHI*).

> **Note:**
>
> When creating LIB files with C++ Builder 5's implib.exe tool, add the "**- a**" option.

12.2.1.3 IIDK Components

The *IIDK* includes the following development tools:

1. *iidk.ocx* – ActiveX control. When installing *Intellect* this file is stored in the Windows\System32 folder and registered in OS.
2. *ddi.exe* – tool used for viewing and editing DDI- and DBI- files. It is stored in the <*Intellect* installation directory>\Tools folder.

12.2.2 Connecting to Intellect

12.2.2.1 Connection Parameters

Intellect's core interacts with functional modules (third party information systems) according to the following connection parameters:

1. Port number.
    a. For the video subsystem: 900.
    b. For the **Interface IIDK** object: 1030.
    c. For **ATM** objects: 1009.

> **Note.**

1030 (IIDK) port can be in use to connect ATMs (ATM) (not only 1009 port) - in this case the **ATM** object will be marked with red cross in the hardware tree. For this the **IIDK Interface object is to be created in the hardware tree**.

2. The IP address of the computer that is running Intellect's core.
3. ID (the connection object ID).

Attention!

To connect to video subsystem (port 900) id is to be more than 1 and must not be the same as id of **IIDK Interface objects created in the system**. To connect to **IIDK Interface object** (port 1030) id is to be the same as one of the object specified in the dialog box of *Intellect* settings.

Note.

If connection to the server (**IIDK Interface object**) from remote computer is required, then there is no need to install *Intellect* on the remote computer, but this computer is to be added to *Intellect* configuration on the server (in the **Hardware** tab of the **System settings** dialog box) and **IIDK Interface object** is to be created under the created **Computer** object. In this case the server address is to be specified in IP parameter of Connect function and ID of specified **IIDK Interface object** is to be specified in ID parameter. Take into account the fact that the **Computer** object corresponding to the remote computer is marked with a red cross in the object tree.

Note:

If an MDL file exists (see Section Creating MDL files), the connection to Intellect's core does not require creating the **IIDK Interface** object in the system. The connection ID passed is an empty string (in other words, the ID is "").

12.2.2.2 IIDK Interface Object

With the **IIDK Interface** object one can manage all the elements of the system. The **IIDK Interface** object is created under the **Computer** object in the *Intellect* object tree.

Note

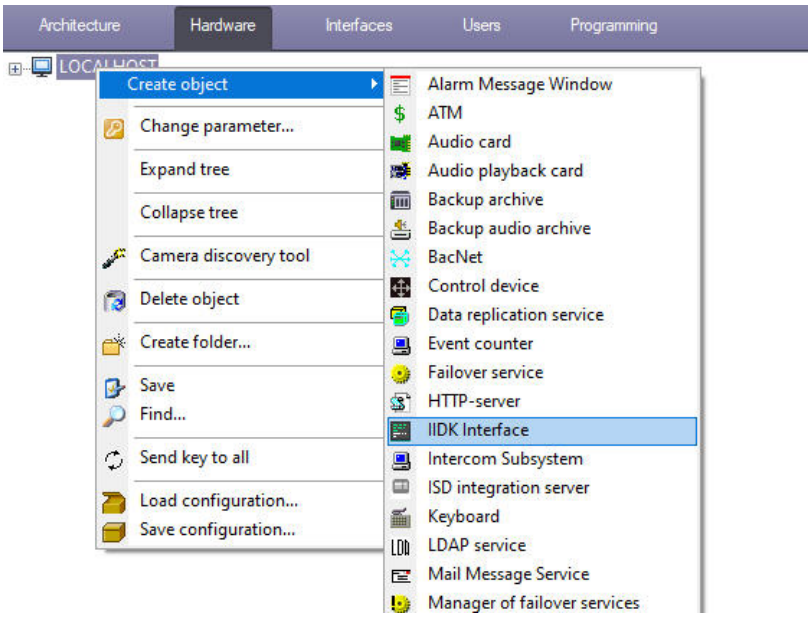
To use the **IIDK Interface** object, allow the relevant functionality in the license key.

Note

If *Intellect* is started in the demo mode, the **IIDK Interface** object is activated after the functional module is connected to the system core (see Connect section)

Important!

The ID of the **IIDK Interface** object must not be the same as the IDs of the **Monitor** objects created in the system.



If the **IIDK Interface** object is used, the settings panels are not created for integrated functional modules (third party applications).

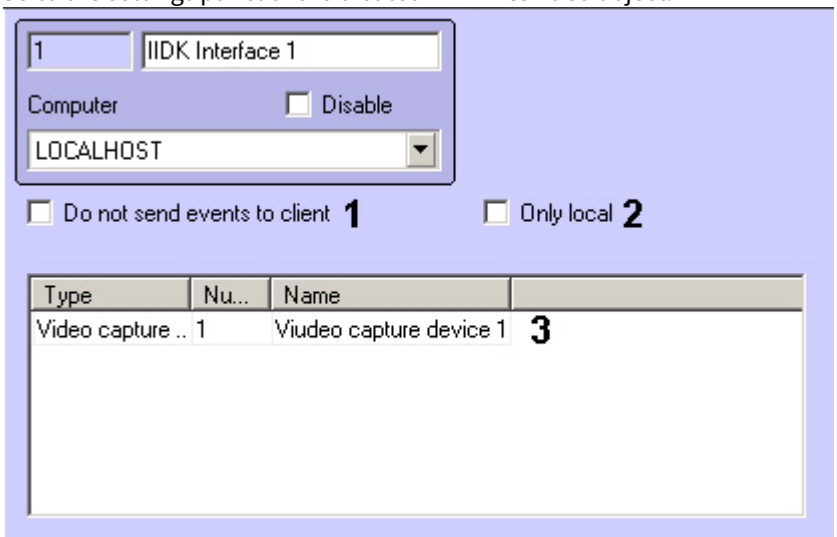
When the *Intellect* distributed architecture is used, the **IIDK Interface** object must be created on the computer that is running the software core (the core to which the connection is made). If the connection is made to a computer that has the *Operator Workstation* installed, the connection parameters must include the IP address of the *Server* or the *Administrator Workstation*.

12.2.2.3 Configuring passing events throung IIDK Interface object

IIDK Interface allows configuring of event filtering passed to connected client applications.

To configure filtering, do the following:

1. Go to the settings panel of the created **IIDK Interface** object.

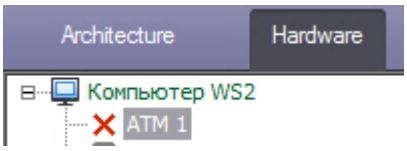


2. Set the **Do not send events to client** if it is not required to send any messages to client application not sending any messages to the core (1).
3. Set the **Only local** checkbox to send to the client applications only messages of those objects created under the same **Computer** object as the **IIDK Interface** object (2).
4. In the table (3) specify list of objects, events from which have to be sent to connected client applications. Enter CORE type to filter the core events.

Configuring of event filtering is completed.

12.2.2.4 Features of ATMs integration. ATM object

The **ATM** object can be used to send events from ATM software to *Intellect* core. This object is created under the **LOCALHOST** object in the **Hardware** tab of the **System settings** dialog box. It is created instead of the **IIDK Interface** object.



The **ATM** object shows ATM events (“Card inserted”, “Withdraw card”, etc.) in the event viewer. These events can be used for captioning, reactions configuration, etc.

> **Note.**
>
> For events regarding client card masked bank card number can be sent in the param0 parameter.

The list of available **ATM** events can be seen using the ddi.exe utility. Information on how to use this utility can be found in the The ddi.exe utility for editing database templates and external settings files section of Administrator's Guide.

Connection method and message syntax for the **ATM** object are the same as those for the **IIDK Interface object**, though 1009 port is in use for sending messages (see also Connection Parameters and Message Syntax (port 900)).

12.2.3 IIDK Functions

12.2.3.1 Connect

To establish communication between a functional module and Intellect, connect to the system core by using the following function:

```
BOOL Connect (LPCTSTR ip, LPCTSTR port, LPCTSTR id, void (_stdcall *func)(LPCTSTR msg))
```

Parameters of the connection function:

Parameter	Description	Example
LPCTSTR ip	The ID address of the computer that is running the system core	
LPCTSTR port	TCP/IP connection port	
LPCTSTR id	A connection ID, for video	
_stdcall *func)(LPCTSTR msg))	A callback function that accepts messages from Intellect	

```
CString port = "900";

CString ip = "127.0.0.1";

CString id = "2";

BOOL IsConnect = Connect(ip, port,
id, myfunc);

if (!IsConnect)

{

  // connection failed

  AfxMessageBox("Error");

}
```

The function returns TRUE if the connection is established, or FALSE if not.

All messages from the system core are accepted by a callback function.

A sample declaration of the callback function:

```
void _stdcall myfunc(LPCTSTR str)
{
 printf("\r\nReceived:%s\r\n\r\n",str);
}
```

Note:

Void _stdcall myfunc is called in a separate stream (not in the application's main stream).

The developer handles received messages as needed.

12.2.3.2 SendMsg

To send messages to the system core, use the following function:

```
BOOL SendMsg (LPCTSTR id, LPCTSTR msg)
```

Parameters of the SendMsg function:

Parameter	Description	Example

LPCTSTR id	The connection ID passed in the call to the **Connect** function	`CString port = "900";`
		
		`CString ip = "127.0.0.1";`
		
		`CString id = "2";`
		
		`BOOL IsConnect = Connect(ip, port, id, myfunc);`
		
		`if (!IsConnect)`
		
		`{`
		
		`// connection failed`
		
		`AfxMessageBox("Error");`
		
		`Return;`
		
		`}`
		
		`SendMsg(id,"CAM|1|REC"); // turn on recording for camera 1`

Parameter	Description	Example
LPCTSTR msg	Message text	`Disconnect (id);`

The function returns TRUE if the message was sent, otherwise FALSE

12.2.3.3 Disconnect

To terminate a connection, use the **Disconnect** function:

```
void Disconnect (LPCTSTR id)
```

, where **LPCTSTR id** is the connection ID passed in the call to the **Connect** function.

If the connection is terminated by Intellect, **DISCONNECTED** is passed to the callback function.

> **Note:**
> An example of using the **Disconnect** function is given in Section SendMsg.

12.2.3.4 Other functions

On the page:

The iidk.h header file contains extra functions that are listed below. The Connect4, SendData4, SendFile and GetMsg functions should not be used. They are created for internal use. The Connect2 function is not used.

12.2.3.4.1 **Connect3**

```
BOOL Connect3(LPCTSTR ip, LPCTSTR port, LPCTSTR id, iidk_callback_func* lpfunc,
              DWORD user_param,int async_connect,DWORD connect_attempts)
```

Parameter	Description
ip	IP address of *Intellect* Server
port	TCP/IP port over which the connection is established

id	Slave connection ID, for video
lpfunc	Callback function receiving messages from *Intellect*
user_param	Extra parameter that comes to Callback function in order to split slaves if there is only one function.
async_connect	0 - synchronous connection mode, the function returns TRUE if the connection is established. -1 - asynchronous connection mode, the function always returns FALSE if the connection is established, then the CONNECTED event is created. Any other value – at first the synchronous mode is used, in case of fault - asynchronous mode.
connect_attempts	Number of connection attempts.

12.2.3.4.2 SendReactToCore

The function is used to send a reaction to the specific core.

```
BOOL SendReactToCore(LPCTSTR id, LPCTSTR msg)
```

Parameter	Description
id	Core connection ID
msg	Messages sent. Message format is similar to SendMsg.

12.2.3.4.3 IsConnected

IsConnected returns TRUE if the client is connected to server.

```
BOOL IsConnected();
```

12.2.3.4.4 Connect4

```
BOOL Connect4(LPCTSTR ip, LPCTSTR port, LPCTSTR id,  iidk_callback_func* lpfunc,
               iidk_frame_callback_func* lpframe_func, iidk_user_data_func*
iidk_user_data_func,
               DWORD user_param,int async_connect,DWORD connect_attempts);
```

Parameter	Description
ip	IP address of *Intellect* Server
port	TCP/IP port over which the connection is *established*
id	Core connection ID, for video
lpfunc	Callback function receiving messages from *Intellect*

lpframe_func	Callback function receiving video frames
iidk_user_data_func	Callback function for data sent using the SendData4 function
user_param	Extra parameter that comes to the Callback function in order to split slaves if there is only one Callback function for all cores.
async_connect	0 - synchronous connection mode, the function returns TRUE if the connection is established -1 - asynchronous connection mode, the function always returns FALSE. If the connection is established, then the CONNECTED event is created Any other value – at first the synchronous mode is used, in case of fault - asynchronous mode.
connect_attempts	Number of connection attempts

12.2.3.4.5 SendData4

This function is used to send CUserNetObject. Its purpose is to send raw data.

```
BOOL SendData4(LPCTSTR id, int nIdent,BYTE *pBuffer,DWORD dwSize);
```

Parameter	Description
id	Core connection ID
nIdent	Data UID
pBuffer	Transmitted data
dwSize	The size of data array

12.2.3.4.6 SendFile

The function is used to send a file.

```
BOOL SendFile(LPCTSTR id, LPCTSTR file_from, LPCTSTR file_to)
```

Parameter	Description
id	Core connection ID
file_from	Address to send file from.
file_to	Address to send file to.

12.2.3.4.7 GetMsg

The function is used to retrieve incoming messages that are queued if the Callback function is not specified.

```
BOOL GetMsg(LPTSTR msg, DWORD& cb)
```

Parameter	Description
msg	Incoming message
cb	Message length

12.2.3.4.8 SetPingTime

The function enables and sets the interval for sending KeepAlive messages to the *Intellect* core. It is enough to call the function once, for example, after calling CreateClient.

```
void SetPingTime(intptr_t clientId, unsigned int time);
```

Parameter	Description
clientId	Client ID
time	The interval for sending KeepAlive messages in milliseconds. The minimum value is 5000; if a smaller value (but not 0) is specified, then the value 5000 will be used. If set to 0, then sending KeepAlive messages is stopped.

12.2.4 Sent Message Syntax

12.2.4.1 Message Syntax

Messages sent to the core have the following syntax:

CORE||DO_REACT|source_type<OBJECT TYPE>,source_id<OBJECT ID>,action<ACTION> [,params<NO. OF PARAMETERS>,param0_name<PARAMETER NAME_0>,param0_val<PARAMETER VALUE_0>]

Below is the syntax of messages that contain two parameters.

CORE||DO_REACT|source_type<OBJECT TYPE>,source_id<OBJECT ID>,action<ACTION>,params<2>,param0_name<PARAMETER NAME_0>,param0_val< PARAMETER VALUE_0>,param1_name<PARAMETER NAME_1>,param1_val<PARAMETER VALUE_1>

The message parameters are described in the table below.

Parameter	Description
source_type<obj>	Object type (see the DDI file ([OBJTYPE] section))
source_id<id>	The object ID set when creating the object in Intellect (see Intellect's settings tree)
action<react>	Action (see the DDI file (the [REACT] section))
params<number>	The number of parameters passed, in decimal format
param0_name<str1>	Parameter name

Parameter	Description
param0_val<str2>	Parameter value

Note:

For working with DDI files, we recommend using the ddi.exe utility (see Section Using the ddi.exe Tool to Work with DDI files).

Example. Sending a message to switch the telemetry to preset mode 4.

CString msg=

"CORE||DO_REACT|source_type<TELEMETRY>,source_id<1.1>,action<GO_PRESET>,params<2>,param0_name<preset>,param0_val<4>,param1_name<*tel_prior*>,param1_val<2>";

SendMsg(id,msg);

12.2.4.2 Message Syntax (port 900)

Messages sent to port 900 are passed to the video subsystem directly; for this reason, such messages have a different syntax.

Messages sent to the video subsystem have the following syntax:

OBJECT TYPE|OBJECT ID|ACTION [|PARAMETER<VALUE>]

Below is the syntax of messages that contain n parameters.

OBJECT TYPE|OBJECT ID|ACTION [|PARAMETER 1<VALUE>,PARAMETER 2<VALUE>,...,PARAMETER N<VALUE>]

Attention!

Port 900 may only be used to manage objects of the GRABBER, CAM, or MONITOR types.

The message parameters are described in the table below.

Parameter	Description
Object type	Object type (GRABBER, CAM, or MONITOR)
Object ID	The object ID set during creation of the object in Intellect
Action	Action (command)
Parameter <Value>	Parameter name. The value is enclosed by angle brackets.

Example 1. Setting camera 1 to recording mode.

CString msg = "CAM|1|REC";

SendMsg (id,msg);

Example 2. Saving video from all cameras to local disk C.

CString msg = "GRABBER|1|SET_DRIVES|drives<C:\>" ;

SendMsg(id,msg);

Note.

The **SET_DRIVES** command includes the ID of any of the video capture cards created in the system.

Note.

The **SET_DRIVES** command does not change the video archiving settings set in the system.

12.2.4.3 Using the Event and React classes

For working with messages, you may use the classes provided: *Event* and *React,* declared in the msg.h file.

Example of using the react class:

A message composed without using the classes	A message composed using the React class
CString msg = "CORE||DO_REACT|source_type<TELEMETRY>,source_id<1.1>, action<GO_PRESET>,params<2>,param0_name<preset>,param0_val<4>, param1_name<*tel_prior*>,param1_val<2>"; **SendMsg(id,msg);**	**React react("TELEMETRY","1.1","GO_PRESET");** **react.SetParamInt("preset",4);** **react.SetParamInt("*tel_prior*",2);** **SendMsg(id,react.MsgToString().c_str());**

Note:

The msg.h and msg.cpp files are located in the Misc folder which is in the archive on page Intellect integration guide (HTTP API, IIDK, ActiveX, HTTP Server, Axxon Next).

12.2.5 Examples of Managing System Objects

12.2.5.1 Adding, Updating, and Deleting System Objects

On the page:

System objects are added, updated, and deleted by the following commands:

1. **CORE||CREATE_OBJECT** – creates a new object.
2. **CORE||UPDATE_OBJECT** – updates an existing object or creates a new one.
3. **CORE||DELETE_OBJECT** – deletes an object.

12.2.5.1.1 Adding a User to a Department

Below is a message that adds the specified user to the specified department, with the specified parameters:

CORE||CREATE_OBJECT|
objtype<PERSON>,objid<12341>,parent_id<1>,surname<Tim>,name<Kovac>,card<12362>,facility_code<0>

IIDK will return the following message in response to this command:

CORE||CREATE_OBJECT|card<1234>,objtype<PERSON>,guid_pk<{281A172C-62D2-EA11-A54B-B06EBF811A34}>,facility_code<122>,surname<Tim>,module<iidk_client_test_x64.exe>,time<16:42:53>,parent_id<1>,fraction<797>,date<30-07-20>,name<Kovac>,owner<QA-T49>,slave_id<QA-T49.11>,objid<12341>

This allows receiving the ID of the created object in the objid<> parameter.

12.2.5.1.2 Adding and Deleting a Video Capture Card

If an object is not present in the system, add that object with the **UPDATE_OBJECT** command (the system must not have an object with a type and ID equal to objtype and objid, respectively).

CORE||UPDATE_OBJECT|objtype<GRABBER>,objid<12>,core_global<0>,parent_id<SLAVAXP>,name<Frame **grabber** 1>,params<5>,param0_name<format>,param0_val<NTSC>,param1_name<mode>,param1_val<1>,param2_name<chan>,param2_val<2>,param3_name<type>,param3_val<FX 4>,param4_name<resolution>,param4_val<0>

Having received the following message, the system changes the name of an existing object:

CORE||UPDATE_OBJECT|objtype<GRABBER>,objid<12>,core_global<0>,parent_id<SLAVAXP>,name<Card **2>**

To delete an object and all of its child objects, use the **DELETE_OBJECT** command:

CORE||DELETE_OBJECT|objtype<GRABBER>,objid<12>

12.2.5.2 Working with the System in the Multiuser Mode

The remote computer must install and be running Intellect (**Client** installation version) in order to exchange messages with the Server.

If users have been created and access rights have been configured in Intellect, any message that requires a response from the system core must contain the **receiver_id<ID>** parameter, where ID is the ID of the **IIDK Interface** in the system.

CORE||GET_CONFIG|objtype<CAM>,objid<1>,receiver_id<1>

// Returns the parameters of the Camera 1 object

To get the user ID and user's permissions DI by username and password, use the CHECK_USER function. Examples of using:
CORE||CHECK_USER|password<1>,login<1>
CORE||CHECK_USER|pass_key<1373503546>,login<1> (crc32 from DB)
CORE||CHECK_USER|md5<bf03b1605e3c83978514f2a6546eef50> (md5 from DB)
Response:
ACTIVEX|1|USER_RIGHTS|rights_id<1>,user_id<1>
If the password is incorrect, the response comes with a delay of 1 second.

12.2.5.3 Determining Computers Where Intellect was Unloaded (via Port 1030)

If Intellect is unloaded, the callback function receives a message with an **action** parameter value of **DISCONNECTED**:

ACTIVEX|12|EVENT|SOCKET<>,MMF<>,objaction<DISCONNECTED>,TRANSPORT_TYPE<MMF>,core_global<1>, action<DISCONNECTED>, module<slave.exe>, objtype<SLAVE>,__slave_id<SLAVAXP.12>, objid<SLAVAXP>,owner<SLAVAXP>,TRANSPORT_ID<1111>,time<12:41:16>,date<23-09-02>

The message contains the name of the computer on which *Intellect* was unloaded and the date and time

12.2.5.4 Redirecting Video Cameras to the Monitor

After receiving the following message, the system deletes all cameras from the monitor and calls the specified video camera:

CORE||DO_REACT|
source_type<MONITOR>,source_id<1>,action<REPLACE>,params<4>,param0_name<slave_id>,param0_val<SLAVA>,param1_name<cam>,param1_val<1>,param2_name<control>,param2_val<1>,param3_name<name>,param3_val<>

If connected via port 900, the above action is performed by using the following message:

MONITOR|1|REPLACE|slave_id<SLAVA>,cam<1>,control<1>

12.2.5.5 Obtaining Object Parameters (via Port 1030) GET_CONFIG

An example of use of the **GET_CONFIG** command is given below:

CORE||GET_CONFIG|objtype<CAM>,objid<1>

The returned message contains all the parameters of the specified object:

ACTIVEX|12|OBJECT_CONFIG|
rec_priority<0>,mask0<>,decoder<0>,mask1<>,flags<>,mask2<>,compression<3>,sat_u<5>,mask3<>,proc_time<>,
hot_rec_period<>,mask4<>,telemetry_id<>,manual<1>,region_id
<1.1>,contrast<5>,md_mode<0>,md_size<5>,audio_type<>,pre_rec_time<0>,config_id<>,
bright<7>,alarm_rec<0>,
audio_id<>,rec_time<>,hot_rec_time<2>,activity<>,mux<0>,parent_id<1>,objtype<CAM>,type<>,__slave_id<SLAVAXP.12
>,objid<1>,
name<Camera 1>,objname<Camera 1>,color<1>,priority<0>,md_contrast<5>

> **Note:**
>
> To obtain the configuration of all the objects of the specified type, remove the **objid** parameter.

Example. Get information about user by the identifier.

CORE||GET_CONFIG|objtype<PERSON>,objid<1>

The response is the message with parameters containing necessary information, including user name, card number etc.:

ACTIVEX|1|OBJECT_CONFIG|
pnet3_sound<0>,galaxy_dual_focus<0>,auto_pass_type<>,galaxy_pin_change<0>,external_id<>,card_date<26.05.2017
10:57:06>,galaxy_tag_link<0>,rubeg8_zone_id<>,levels_times<>,expired<>,hid_escort_id<>,objtype<PERSON>,level2_id<>,galax
y_group_choice<0>,
who_level<>,hid_use_extended_access<0>,visit_purpose<>,**card<1234>**
,email<>,galaxy_timer_schedule<0>,galaxy_menu_option<0>,aiu_holiday<0>,
area_id<>,aiu_alarm<0>,objname<User 1>,surname<>,who_card<>,auto_brand<>,pnet3_alarm<0>,card_loss<0>,
facility_code<432>
,galaxy_temp_code<0>,post<>,when_area_id_changed<>,drivers_licence<>,bolid_in_device<0>,pnet3_acs_counter<0>,
temp_levels_times<>,temp_card<>,pnet3_no_entry<0>,location<>,temp_level_id<>,patronymic<>,teleph_work<>,department<
>,galaxy_keypad<0>,
_TRANSPORT_ID<>,finished_at<>,aiu_ksd_type<>,all_param<>,galaxy_template<0>,tabnum<>,parent_id<1>,pur<>,galaxy_dures
ss<0>,pnet3_no_exit<0>,
galaxy_dual<0>,hid_pin_exempt<0>,pnet3_counter<0>,whence<>,schedule_id<>,hid_enable_pin_commands<0>,galaxy_dual_a
ccess<0>,
pnet3_block<0>,passport<>,person<>,galaxy_menu_choice<0>,flags<0>,auto_number<>,phone<>,pin<>,rubeg8_AccessToBCPTi
meZoneNumber<>,
begin_temp_level<>,pnet3_master<0>,aiu_mark<0>,end_temp_level<>,visit_birthplace<>,galaxy_menu_level<>,visit_document
<>,pnet3_guard<0>,
pnet3_black<0>,aiu_kso_type<>,is_apb<0>,**name<User 1>**,pnet3_temp<0>,started_at<>,level_id<>,_marker<>,
bolid_user_type<>,owner_person_id<>,card_type<>,is_active_temp_level<0>,begin<>,hid_line_tag<>,guid<{5B358685-E041-
E711-BCC6-DA0AE28E0C17}>,
pnet3_guest<0>,is_locked<0>,objid<1>,marketing_info<>,comment<>,aiu_vpu_arm<0>,visit_reg<>,bolid_in_pku<0>,pnet3_2car
ds_mask<0>

> **Note.**
>
> User data can also be received via direct request to *Intellect* software database from the OBJ_PERSON table. In this case you can select a user by card number or other parameters. See more info on *Intellect* software database operation in Administrator's Guide, the Appendix 4. INTELLECT™ software database management.

12.2.5.6 Obtaining Information on Object States GET_STATE and GET_LIST

To obtain information on the state of an object, use the **GET_STATE** command:

CORE||GET_STATE|objtype<CAM>,objid<1>

The following string is returned:

ACTIVEX|12|OBJECT_STATE|objtype<CAM>,__slave_id<SLAVAXP.12>,objid<1>,state<DISARM_DETACHED>

The state of the specified object is represented by the **state** parameter, which takes values from the set of states that are specified in the object's DDI file.

If connected via port 900, requests for object states are performed through the **GET_LIST** command:

CAM||GET_LIST

> **Note:**
>
> Regardless of whether an object ID is specified, the command returns the states of all objects of the specified type.

The returned message has the following format:

CAM|1|SETUP|rec_priority<0>,is_armed<0>,is_recorded<0> , bt<0> , slave_id<SLAVAXP>, compression<3>,sat_u<5>, proc_time<0>, hot_rec_period<0> , manual<1>, telemetry_id<>, is_detached<1> , contrast<5>, md_size<5>,md_mode<0>, is_alarmed<0>, audio_type<>, pre_rec_time<0>, bright<7>, audio_id<>, rec_time<0>, alarm_rec<0>, hot_rec_time<2>, mux<0>, parent_id<1>, __slave_id<SLAVAXP>, priority<0>, mask<>, color<1>,md_contrast<5>, is_ring<1>, fs_error<0>

The message presents the states as follows: **is_state<val>,** where **state** is an object state (see the DDI file); and **val** equals 1 if the object is in this state, 0 otherwise.

> **Note.**
>
> The **is_ring<>** parameter shows whether loop recording is performing or not. The **fs_error** parameter equals 1 when there was an archive recording error (e.g. failed to delete folder for loop recording).

12.2.5.7 Showing Information Messages. SET_STATE

To show an information message on the display of Intellect's main control panel, use the SET_STATE command:

CORE||SET_STATE|name<POS 1>,value<Can’t open port COM4>

The figure below shows the result of handling the message by the system.

The message is removed from the display as follows:

CORE||SET_STATE|name<POS 1>,value<>

12.2.5.8 Live and archived video

12.2.5.8.1 Get live video

To get live video from Camera 1 send the following message to port 900:

CAM|1|START_VIDEO|compress<1>

Here compress<> is compression ratio, from 0 to 5. Video frames will be received as a response to this message. The example of how to process incoming frames is given in demo kit available for download at the page of Intellect integration guide (HTTP API, IIDK, ActiveX, HTTP Server, Axxon Next).

12.2.5.8.2 Get archived video

To get archived video from Camera 1 send the following messages to port 900:

CAM|1|ARCH_FRAME_TIME|time<dd-mm-yy HH:MM:SS.FFF> (to specify start time for viewing the archive) or **CAM|1|PLAY|compress<>** (to get archived video. Archived video is handled the same way as live video)

12.2.5.8.3 Get the list of time intervals

In order to get the full list of time intervals with video recordings for exact date, send the following message to port 900:

CAM|id|ARCH_GET_INTERVALSREC|date<>,time_with_milliseconds<1>,with_filenames<1>

The date<> parameter can take the date<dd-mm-yy|> value or it can be left blank. In the first case, the intervals for the specified date will be requested, in the second case, the dates for which the archive is present will be requested..

If the time_with_milliseconds<1> parameter is set, then the intervals will be received with milliseconds (for example, 14:08:55.**677** 14:09:55.**641**). If the with_filenames<1> parameter is set, then the message will contain filenames (for example, 'C:\VIDEO\26-04-21 14\0._01'). These parameters are optional.

As a result the following message is received:

Event: CAM|id|SET_INTERVALSREC|intervals<>,date<>,timezone<>

The value of intervals<> parameter looks like this: intervals<begin1 end1\nbegin2 end2…\nbeginN endN|date1\ndate2…\ndateN\n>

The time of beginning and ending are one blank separated (0x20 code), intervals are line break separated '\n'(0x0A code).

- begin1, begin2, ... beginN – time of interval beginnings in the HH:MM:SS format (returns if the exact date was requested).
- end1, end2, ... endN – time of interval endings in the HH:MM:SS format (returns if the exact date was requested).
- date1, date2, ... dateN – dates at which there are recordings in the archive (returns if the date field in the request is blank or there is no such field).

date<dd-mm-yy|> parameter represents date for which the intervals were requested or blank value (date<>) if dates for the entire period were requested.

timezone<> shows time shift on the client (IIDK) relative to Server time, in minutes. Examples:

- If Server is in -1 UTC time zone, the response to the CAM|1|ARCH_GET_INTERVALSREC| command will have parameter timezone<60>
- If Server is in +3 UTC time zone, the response to the CAM|1|ARCH_GET_INTERVALSREC| command will have parameter timezone<-180>

Request example:

```
CAM|1|ARCH_GET_INTERVALSREC|date<26-04-21>,time_with_milliseconds<1>,with_filenames<1>
```

Response example:

```
CAM|1|SET_INTERVALSREC|time_with_milliseconds<1>,intervals<14:07:55.659 14:08:55.637 - -
file:'C:\VIDEO\26-04-21 14\0._01'
14:08:55.677 14:09:55.641 - - file:'C:\VIDEO\26-04-21 14\1._01'
14:09:55.681 14:10:35.667 - - file:'C:\VIDEO\26-04-21 14\2._01'
14:17:12.444 14:18:09.553 - - file:'C:\VIDEO\26-04-21 14\3._01'
14:29:41.132 14:29:41.292 - - file:'C:\VIDEO\26-04-21 14\4._01'
14:29:41.432 14:29:51.363 - - file:'C:\VIDEO\26-04-21 14\5._01'
14:34:38.788 14:35:03.117 - - file:'C:\VIDEO\26-04-21 14\6._01'
14:35:03.267 14:36:19.151 - - file:'C:\VIDEO\26-04-21 14\7._01'
>,timezone<-180>,module<iidk_client_test_x64.exe>,_TRANSPORT_ID<>,with_filenames<1>,date<26-04-
21>,slave_id<VDESKTOP.2A180FE5-BA8E-420C-B80E-90DC20516A26>,durationMS<0>
```

12.2.5.9 Telemetry control via IIDK

Telemetry is controlled via IIDK using simple reactions described in the TELEMETRY section of Programming guide, for instance:

CORE||DO_REACT|source_type<TELEMETRY>,source_id<1.1>,action<LEFT>,params<1>,param0_name<tel_prior>,param0_val<3> – message sent to port 1030 in order to rotate camera lens left with high priority.

TELEMETRY|1.1|LEFT|speed<2>,tel_prior<3> – reaction to port 1030 in order to rotate camera lens left with high priority at an average speed.

12.2.5.10 Map layer operations

The command for setting the parameter and position of **Camera 1** object icon is run in one of the following ways:

1. Sending a message to port 1030 **CORE||DO_REACT|source_type<MAPLAYER>,source_id<1>,action<CUSTOMIZE_OBJECT>,params<7>,param0_name<x>,param0_val<200>,param1_name<y>,param1_val<200>,param2_name<objtype>,param2_val<CAM>,param3_name<objid>,param3_val<1>,param4_name<a>,param4_val<90>,param5_name<w>,param5_val<70>,param6_name<h>,param6_val<80>**
   Where *x*, *y*, *w* and *h* are the coordinates and size of the object icon on the map.
   a is a tilt angle of icon.
2. Sending a reaction to port 1030 **MAPLAYER|1|CUSTOMIZE_OBJECT|x<200>,y<200>,objtype<CAM>,objid<1>,a<90>,w<70>,h<80>**

Layer 1 is shown in the interactive map window using one of the following ways:

1. Sending a message to port 1030: **CORE||DO_REACT|source_type<MAPLAYER>,source_id<1>,action<ACTIVATE>**
2. Sending a reaction to port 1030: **MAPLAYER|1|ACTIVATE**

12.2.5.11 Obtaining info on core queues with GET_QUEUE_INFO command

Use the GET_QUEUE_INFO command to request info about the queues in the Intellect core:

CORE||GET_QUEUE_INFO

> **Note.**
>
> The receiver_id parameter may be specified if there are several **IIDK Interface** objects in the system – see Working with the System in the Multiuser Mode.

The response is the string like this:

ACTIVEX|11|QUEUE_INFO|thread2<0>,thread1<0>,thread0<0>,posted_events<0>,_TRANSPORT_ID<>,server_reacts<0>,posted_reacts<0>,events_inwork<0>,coremanager_events<0>,thread3<0>

The response parameters correspond to the information displayed in the **Queue statistics** tool (shows on Alt+F2). Parameters description:

threadN<> – number of items in the queue of the thread N.

posted_events<> – number of incoming events.

posted_reacts<> – number of reactions currently being processed.

coremanager_events<> – number of events to send.

server_reacts<> – number of reactions to send.

events_inwork<> – number of events currently being processed.

12.2.5.12 Playing audio archive for a period. START_PLAY_TIME command

To play the audio archive of a specified microphone for a specified period using a specific **Speaker** object, execute a command like

SPEAKER|{id}|START_PLAY_TIME|speaker_id<>,mic_id<>,time_start<>,time_end<>,cam_id<>

Examples:

SPEAKER|1|START_PLAY_TIME|speaker_id<1>,mic_id<2>,time_start<28-02-20 14:23:24.092>,time_end<28-02-20 14:23:27.092>

SPEAKER|1|START_PLAY_TIME|speaker_id<1>,mic_id<2>,time_start<02-03-20 12:01:24.092>,time_end<02-03-20 12:04:27.092>,cam_id<1>

If camera ID **cam_id<>** is specified, the audio archive recorded synchronously with the video archive (i.e. from the VIDEO folder) will be played. If the **cam_id<>** parameter is not specified, then the archive from the AUDIO folder is played.

speaker_id<> – identifier of the **Speaker** object

mic_id<> – identifier of the **Microphone** object

time_start<> – start time of the audio archive segment

time_end<> – end time of the audio archive segment

12.2.5.13 Playback of the dial tone. The START_TONE and STOP_TONE commands

To playback the numbers, letters, and symbols in a dial tone using a specific **Speaker** object, run the START_TONE command:

```
SPEAKER|{id}|START_TONE|speaker_id<>,symbols<>,duration<>
```

Command parameters:

- **speaker_id<>**—identifier of the Speaker object on which you want to perform the playback;
- **symbols<>**—symbols that should be played back. Only 1234567890*#ABCD can be symbols, others will be ignored. The letter case doesn't matter;
- **duration<>**—duration of sounding of each symbol in milliseconds (ms).

To stop the playback early, run the STOP_TONE command:

```
SPEAKER|{id}|STOP_TONE|speaker_id<>
```

Here, **speaker_id<>**—identifier of the Speaker object on which you want to stop the playback.

Command examples:

```
SPEAKER|1|START_TONE|speaker_id<1>,symbols<12>,duration<200>
SPEAKER|1|STOP_TONE|speaker_id<1>
```

12.2.5.14 Get statistics on the video stream. STATISTIC

You can use the following commands to display video stream statistics:

- **STATISTIC||GET**—get video statistics;
- **STATISTIC||START|interval<>**—start sending statistics at an interval specified in seconds (for example, interval<10>);
- **STATISTIC||STOP**—stop sending statistics.

The commands are sent through 900 port. Examples of the received messages with statistics:

STATISTIC|[MONITOR][2][CAM][9][IN]|SET|count<20>,bps<1.42042e+007>,fps<19.728>,_TRANSPORT_ID<>
STATISTIC|[MONITOR][2][CAM][9][OUT]|SET|count<20>,bps<888264>,fps<19.7392>,_TRANSPORT_ID<>

13 CamMonitor.ocx ActiveX Control

13.1 General description of CamMonitor.ocx component of ActiveX

On the page:

13.1.1 General information

CamMonitor.ocx is the component of ActiveX that is similar in every way to the **Video monitor** interface object. It allows you to manage cameras, view the archive, etc.

CamMonitor.ocx component supports operation in the demo mode.

13.1.2 Requirements to developers

To use CamMonitor.ocx you will need:

1. The knowledge of any programming language that supports using the Component Object Model (COM);
2. Basic knowledge of Win32/Win64 programming;
3. Programming environment that supports OCX files.

Requirements for software which is used while integrating

13.2 How to install CamMonitor.ocx

Important!

It is not recommended to install *Intellect* and CamMonitor.ocx on the same computer. If it is required, then their versions are to be the same, otherwise CamMonitor.ocx operation is not guaranteed.

Use the CamMonitorInstaller.exe file (stored in the <*Intellect* installation directory>\Redist\CamMonitor folder) to install CamMonitor.ocx. Both 32-bit and 64-bit versions of this component are available. The installation files of components of different bit count are stored in the corresponding folders (x86 and x32 correspondingly).

If *Intellect* software is installed on the computer, then the CamMonitor.ocx file is stored in the <*Intellect* installation directory>\Modules\ folder when installing the 32-bit version and it is stored in the <*Intellect* installation directory>\Modules64\ folder when installing the 64-bit version.

If *Intellect* software is not installed, then the 32-bit component is installed in the C:\Program Files (x86)\ITV VideoPlayer\Modules\ folder and the 64-bit component is installed in the c:\Program Files\ITV VideoPlayer x64\Modules64\ folder.

There is standard registration of CamMonitor.ocx as ActiveX component at the stage of installation.

CamMonitorInstaller.exe installs the required files for all users.

Besides the library itself, CodecPack driver pack and ITV VideoPlayer utility are installed. ITV VideoPlayer utility uses the CamMonitor.ocx component and enables viewing the archive from the selected camera. By default this utility is installed in the C:\Program Files\ITV VideoPlayer\ folder. The utility interface looks like one of Converter.exe, but some features of Converter.exe

(not dealing with viewing the archive) are not available. This utility can be used to check if CamMonitor.ocx is installed correctly.

13.3 CamMonitor.ocx parameters

On the page:

The parameters used for setting the CamMonitor component are presented in this section: set the display elements of the interface, as well as the overlay mode.

All parameters are *long* integers.

The values of the parameters used for interface setup are listed in the tables and formed in a way that there is only one unit in the binary representation of the number. To set the value of a parameter, combine the values of parameters using the XOR operation. You'll get the number the positions of which in binary representation indicates which interface elements should be displayed and which should be hidden. See How to use parameters.

The OverlayMode parameter differs from others: it takes values from 0 to 2, and its value sets the overlay mode.

13.3.1 CamMenuOptions

CamMenuOptions : long

Allows setting the feature menu of the camera.

One or more checkboxes can be set.

Available values:

Value	Information
#define MENU_ENABLE_OPTION 0x00000001	Deprecated. See BUTTON_MENU_ENABLE_OPTION
#define MENU_ARM_ENABLE_OPTION 0x00000002	Show the **Arm** option
#define MENU_REC_ENABLE_OPTION 0x00000004	Show the **Start recording** option
#define MENU_CAMS_ENABLE_OPTION 0x00000008	Show the **Camera** option
#define MENU_TITLES_ENABLE_OPTION 0x00000010	Show the **Show titles** option
#define MENU_PROCESSING_ENABLE_OPTION 0x00000020	Show the **Processing** option
#define MENU_EXPORT_ENABLE_OPTION 0x00000040	Show the **Export** option

13.3.2 CamMenuProcessingOptions

CamMenuProcessingOptions : long

Allows setting the **Processing** menu in the feature menu of the camera.

One or more checkboxes can be set.

Available values:

Value	Information
#define MENU_PROCESSING_DEINTERLACE_ENABLE_OPTION 0x00000001	Show the **Deinterlacing** option
#define MENU_PROCESSING_ZOOM_ENABLE_OPTION 0x00000002	Show the **Zoom-in** option *Note. If this option is disabled, the **Processing** → **Zoom** menu option is not shown, but zooming with mouse wheel is still available*
#define MENU_PROCESSING_CONTRAST_ENABLE_OPTION 0x00000004	Show the **Contrast** option
#define MENU_PROCESSING_MASK_ENABLE_OPTION 0x00000008	Show the **Detector mask** option
#define MENU_PROCESSING_SHARP_ENABLE_OPTION 0x00000010	Show the **Sharpen** option

13.3.3 CamButtonsOptions

CamButtonsOptions : long

Allows setting the display of the CamMonitor component buttons.

One or more checkboxes can be set.

Available values:

Value	Information
#define BUTTON_MODE_ENABLE_OPTION 0x00000100	Show the Archive button
#define BUTTON_TIME_ENABLE_OPTION 0x00000002	Show time
#define BUTTON_NAME_ENABLE_OPTION 0x00000004	Show camera name
#define BUTTON_MENU_ENABLE_OPTION 0x00000008	Show the Menu button
#define BUTTON_RAYS_ENABLE_OPTION 0x00000010	Not used
#define BUTTON_MICS_ENABLE_OPTION 0x00000020	Not used
#define BUTTON_ARCH_PANEL_ENABLE_OPTION 0x00000200	Show the archive navigation elements

13.3.4 MainPanelOptions

MainPanelOptions : long

Allows setting the display of the CamMonitor panel.

One or more checkboxes can be set.

Available values:

Value	Information
#define MAIN_PANEL_ENABLE_OPTION 0x00000001	Show the panel
#define MAIN_PANEL_ENABLE_SCREENS_BUTTON 0x00000010	Show the **Screens** button (see Windows layout on the monitor)
#define MAIN_PANEL_ENABLE_BOOKMARK_BUTTON 0x00000020	Show the **Create a bookmark** button (see Create a bookmark)
#define MAIN_PANEL_ENABLE_BOOKMARK_REVIEW_BUTTON 0x00000040	Show the **List of bookmarks** button (see List of bookmarks)
#define MAIN_PANEL_ENABLE_AVIEXPORT_BUTTON 0x00000080	Show the **Background export** button (see The AviExport utility)

13.3.5 KeysOptions

KeysOptions : long

Allows setting the control over the component using the keyboard and the mouse.

One or more checkboxes can be set.

Available values:

Value	Description
#define KEYS_ENABLE_OPTION 0x00000001	Enables control over the CamMonitor component using the hotkeys available for Video Monitor (see Video surveillance monitor)
#define TELEMETRY_DISABLE_OPTION 0x00000002	Disables Telemetry control using the CamMonitor component (see Telemetry control)
#define ARCH_DELETE_ENABLE_OPTION	Enables archive recordings deletion from the recordings list (see Deleting video recordings from the archive)
#define ARCH_PROTECT_ENABLE_OPTION	Enables rewrite protection of the archive recordings from the recordings list (see Protection of separate record and disable of protection)

13.3.6 OverlayMode

OverlayMode : long

Sets the overlay mode.

Available values:

Value	Information
0	Overlay is not in use
1	Overlay 1
2	Overlay 2

13.3.7 How to use parameters

```
DWORD options = CamMonitor1->CamMenuOptions;
options = options^MENU_CAMS_ENABLE_OPTION^MENU_ARM_ENABLE_OPTION^MENU_REC_ENABLE_OPTION;
CamMonitor1->CamMenuOptions = options;
CamMonitor1->CamMenuProcessingOptions ^= MENU_PROCESSING_MASK_ENABLE_OPTION;
```

13.4 CamMonitor.ocx methods

On the page:

13.4.1 Connect

Connect(BSTR **ip**, BSTR **login**, BSTR **password**, BSTR **arch_password**, long **param**, long **port**) set up a connection to the Server/ Video Gate/Backup Archive.

- BSTR **ip** – IP address of the Video server;
- BSTR **login** – login to connect to the Server (can be blank);
- BSTR **password** – password to set up a connection to the Video server (can be blank);
- BSTR **arch_password** – password to access the archive (i.e. admin password, can be blank);
- long **param** – Server role. The parameter is mandatory.
  - 0 – video server;
  - 1 – backup archive;
  - 2 – videogate;
- long **port** – port to connect Video server.
  - if 0, 1 or 2 are passed, the connection is established with port 900, 901 or 902 correspondingly;
  - if 100 is passed, the connection is with port 10504;
  - if any onter value passed, the connection is with port number "port + 20000". For example, if port=900, the connection is established with server port 20900.

The connection to Server is set up **asynchronously**.

Important!

If login and password are not specified at Connect() method call, all cameras are viewable in the control. It is to be considered when developing third-party application if access privileges are relevant.

13.4.2 ShowCam

ShowCam(long **cam_id**, long **compress**, long **show**) shows/hides camera on the monitor.

- long **cam_id** – camera ID
- long **compress** – level of video compression 0-5 (for local camera =0). If set to -1, video stream is displayed directly from the camera without compression.
- long **show** – checkbox: show/hide camera (1/0)

13.4.3 DoReactMonitor

DoReactMonitor(BSTR **react_string**) – control over the monitor/cameras

- BSTR **react_string** – reaction string view

How to create react_string:

```
react_string =   "MONITOR||ARCH_FRAME_TIME|cam<3>,date<dd-mm-yy>,time<hh:mm:ss>";
CamMonitor1->DoReactMonitor(react_string);
```

The result of calling the function with the parameter: camera 3 will be in the archive mode and the archive will be positioned to date «dd-mm-yy» and time «hh:mm:ss» (date and time are to be set in this format only).

The mode parameter takes the following values:

0 – video gate if it's specified (otherwise, video server).

1 – video server.

2 – long-time archive.

“MONITOR|**<id ignored>**|ARCH_FRAME_TIME|...”

> **Note:**
>
> Positioning accuracy can be specified in milliseconds, for instance:
> `DoReactMonitor(“MONITOR||ARCH_FRAME_TIME|cam<3>,date<02-10 05>,time<12:12:22.345>`

Example. Show 2nd stream from camera 14 on Monitor 1:

"MONITOR|1|ADD_CAM|cam<14>,cam_id<14>,compress<1>,stream_id<14.2>"

Exapmle. Set fps = 1 when viewing archive from Camera 11

"MONITOR||CAM_PARAMS|cam<11>;arch_fps<1>"

13.4.4 RemoveAllCams

RemoveAllCams() : long – remove all cameras from the monitor

13.4.5 IsConnected

IsConnected() : boolean – method shows if the Video server is connected or disconnected.

13.4.6 GetCurIp

GetCurIp() : BSTR – returns the IP address of Server specified when calling **Connect**.

13.4.7 SendRawMessage

SendRawMessage(BSTR **msg**) – sends the command to be executed to Video server.

- BSTR **msg** – command string view

How to call a function:

```
m_Cam.SendRawMessage("CAM|1|REC");
```

```
m_Cam.SendRawMessage("CAM|1|REC_STOP");
```

```
m_Cam.SendRawMessage("CAM|1|ARM");
```

```
m_Cam.SendRawMessage("CAM|1|DISARM");
```

13.4.8 Disconnect

Disconnect() – disconnect from the Video server.

13.4.9 SetCallBackOptions

SetCallBackOptions(int **cam_id**, int **options**) – sets parameters of getting video from camera.

- int **cam_id** – camera ID (number).
- int **options** – options. The possible values of the **options parameter are:**

- WithoutVideoFrame = 0x00 – do not send frames from the video module.
- WithVideoFrame = 0x01 – send frames from the video module.
- WithExtendedParams = 0x02 – get frames with extended parameters (time, fps, subtitles).
- WithInformationLayout = 0x04 – display video in the window with control elements (context menu).
- WithCompressedData = 0x08 – display video in the native format without decompression (if any).

- WithoutDecode = 0x10 – disable video decoding on the server.
- WithoutSubtitles=0x20 – disable subtitles.

> **Note**
>
> The options parameter is created the same as parameters of the CamMonitor.ocx component – see CamMonitor.ocx parameters.

13.4.10 SetParam

SetParam(BSTR **param_name**, BSTR **param_value**) - sets the number of camera windows in CamMonitor.

- BSTR **param_name** - a string representation of the length or width.
- BSTR **param_value** - the number of camera windows.

How to call a function:

```
m_cam.SetParam("monitor_ch", m_NH);
m_cam.SetParam("monitor_cw", m_NW);
```

13.5 CamMonitor.ocx events

OnCamListChange (long **cam_id**, long **action**) – occurs when there is connection with the Server or the number of cameras on the Server changes.

• long **cam_id** – camera ID.

• long **action** equals 1, if camera with **id** == **cam_id** exists, otherwise **action** == **0.**

This event occurs as many times as there are cameras on the Server. The negative value of the **cam_id** parameter (cam_id < 0) shows that **OnCamListChange** is not called.

If there are 3 cameras (1, 2, 3) on the Server, then the following events will occur one after another:

CamListChange(1,1)

CamListChange(2,1)

CamListChange(3,1)

CamListChange(-1,1)

Example:

Show the camera with cam_id =2 with **compress =1** compression level;

```
CamMonitor1CamListChange(long cam_id, long action)
{
    if(cam_id == -1)
    {
        CamMonitor1->ShowCam(2,1,1);
    }
}
```

14 Intellect HTTP API

14.1 General information on HTTP API

HTTP API is represented by web2 module (*Web-server 2.0*).

> **Note.**
>
> See Administrator's Guide, Configuring the Server to connect the Clients via the Web-server 2.0 module section.

HTTP API allows the following features:

1. Get information about interactive maps: map list, map name, map layer list, layer parameters, layer background image, information about the list of points and an individual point on the layer (see Map).
2. Get information about object classes created on the Server, a list of states for the object class and information about status, icons for a specific state (see Object classes).
3. Get a list of objects created on the server, information about the individual object, the state of the object, the list of available actions with the object (see Objects).
4. Receive events from the Server both separately and by blocks (see Getting events).
5. Send commands to the server (see Sending commands to server).
6. Run macros (see Macros).
7. Work with video: get frames, request configuration, receive live and archive video, manage recording, arm and disarm cameras, manage telemetry (see Video).
8. Get live and archive sound (see Sound).
9. Send events and reactions to the core of the Intellect software (see Sending reactions and events to Intellect using HTTP request).

The following notation is used in the examples shown in this section:

- Port stands for port number. The default **Web-server 2.0** module port is 8085. Specifying port in HTTP API commands is mandatory.
- /web2 – web context where the web2 app operates. This is the web-app context.

Further the description will be omitted when query action is clear in the context.

> **Important!**
>
> URL, id of objects and file extension are case-sensitive.

> **Note.**
>
> Date and time are specified in RFC 3339 format, see details at http://www.ietf.org/rfc/rfc3339.txt

14.1.1 Authorization

Authorization is needed for requests. Two types of authorization are supported: Basic and Bearer.

With the Basic authorization type, it is necessary to add user data to all HTTP requests in the following form:

```
http://[username]:[password]@[IP-address]:[port]/web2
```

With the Bearer authorization type, the token received from the web server is used (see Authorization using a token key).

14.1.2 Default response format

By default, the response is in the JSON format. The default response in XML format can be enabled on the settings panel of the **Web server 2.0** object (see Configuring default response type for HTTP API requests). Also, the response format can be explicitly

specified in the **Accept** header, for example **application/json** or **application/xml**. The response format specified in the request has a higher priority than the default response format specified on the settings panel of the **Web server 2.0** object.

14.1.3 Cross domain requests (CORS)

To perform cross-domain requests or to access the necessary headers in the response (for example, due to CORS browser policy restrictions), it is necessary to specify **Origin** (the domain of the site from which the request is made) in the request header. In this case, the response will contain the **Access-Control-Allow-Origin** header, which indicates that the resource can be accessed from the specified domain in a cross-site manner. The **Access-Control-Allow-Origin: *** header indicates that the resource can be accessed from any domain in a cross-site manner.

14.2 Product version

14.2.1 General request format:

http://IP-address:port/web2/product/version

14.2.2 Request example:

GET http://127.0.0.1:8085/web2/product/version

14.2.3 Response example:

The response is text/plain line like:

```
Intellect/4.11.2.2875
```

It means that the server supports the protocol described in this document. The line can change depending on the product version. This helps to distinguish 2 web servers with similar functionality but different protocols in different products.

14.3 Authorization using a token key

Authorization in *Intellect* using a token key provides the following capabilities:

- To specify a token in the "token" parameter in a url request instead of specifying the "login" and "password" parameters. Example of a video request with authorization in *Intellect* using a token key:

  ```
  http://127.0.0.1:80/video/action.do?
  normalize=true&version=4.10.0.0&video_in=CAM:1&token=EoHWC_zXFILImB0hL4QgjPc5624cJXMF
  ```

- To use the Bearer Token Authentication in the "Authorization" parameter in the request header.
  Example:

  ```
  Authorization: Bearer PJ_eHSwUsqjXX7PRZMB8hm_zKEnCg3hE"
  ```

14.3.1 General format of request:

GET/POST http://{login}:{password}@IP-address:port/token?expires_in={expires_in}

14.3.2 Request parameters:

Parameter	Is required	Description
login	No	User login in *Intellect*, if specified
password	No	User password in *Intellect*, if specified
expires_in	No	Token validity time in seconds. The maximum value is 1 day. The token expires after a specified period of time. The default value is **1800**. To log out, specify the value **0**. *Note. There can be only 1 token for each user.*

14.3.3 Request example:

```
GET/POST http://USER:PASSWORD@127.0.0.1:80/token?expires_in=1800
```

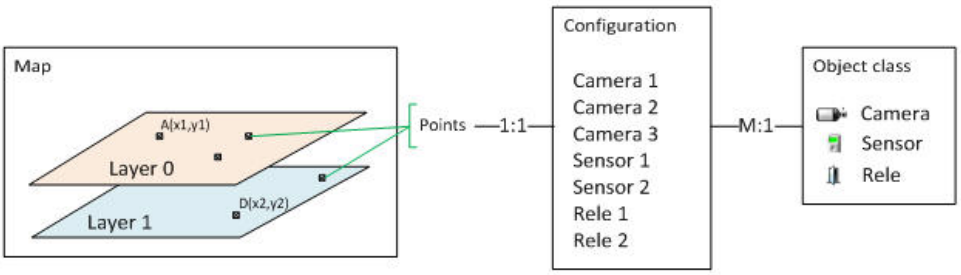
14.3.4 Response example:

```
{
"access_token":"PJ_eHSwUsqjXX7PRZMB8hm_zKEnCg3hE"
"token_type":"bearer"
"expires_in":"1800"
}
```

14.3.5 Response parameters:

Parameter	Description
access_token	Token
token_type	Token type
expires_in	Token validity time in seconds

14.4 Maps



Several maps can be created on the server. Each map can consists of one or more layers. There are points on each layer. Each point corresponds to one of the objects in configuration.

Configuration – objects in Intellect. Each object represents the object of specific class. Each object has one state and the list of actions to be performed.

The object class describes its icons, possible states and possible actions with the object in each state.

14.4.1 Getting the list of maps

14.4.1.1 General request format:

GET http://IP-address:port/web2/secure/kartas/

14.4.1.2 Request example:

GET http://127.0.0.1:8085/web2/secure/kartas/

14.4.1.3 **Response example:**

```
<kartas>
   <karta>
      <id>2</id>
      <layers>
         <layer>
            <geo_h>10.0</geo_h>
            <height>578</height>
            <id>2</id>
            <lat_bl>43.5905</lat_bl>
            <lat_br>43.6875</lat_br>
            <lat_c>43.6395</lat_c>
            <lat_tl>43.5914</lat_tl>
            <lat_tr>43.6885</lat_tr>
            <lon_bl>43.4584</lon_bl>
            <lon_br>43.4599</lon_br>
            <lon_c>43.4809</lon_c>
            <lon_tl>43.5018</lon_tl>
            <lon_tr>43.5033</lon_tr>
            <mapId>2</mapId>
            <name>Layer 2</name>
            <points>
               <point>
                  <id>CAM:1</id>
                  <layerId>2</layerId>
                  <mapId>2</mapId>
                  <angle>0.0</angle>
                  <geo_angle>0.0</geo_angle>
                  <latitude>43.47727</latitude>
                  <longitude>43.602381</longitude>
                  <x>95.0</x>
                  <y>329.0</y>
               </point>
            </points>
            <width>800</width>
            <zoomDef>1.0</zoomDef>
            <zoomMax>4.0</zoomMax>
            <zoomMin>0.25</zoomMin>
            <zoomStep>0.25</zoomStep>
```

```
        </layer>
      </layers>
      <name>Map 2</name>
   </karta>
</kartas>
```

14.4.1.4 Response parameters

Parameter	Description
<karta> group parameters	
id	Map ID
name	Map name
layers	Layer list
<layer> group parameters	
geo_h	Height mark (see Configuring map binding to coordinate grid)
height	Layer substrate height in pixels
width	Layer substrate width in pixels
id	Layer ID
lat_bl	Latitude: bottom left corner
lat_br	Latitude: bottom right corner
lat_c	Latitude: center
lat_tl	Latitude: top left corner
lat_tr	Latitude: top right corner
lon_bl	Longitude: bottom left corner
lon_br	Longitude: bottom right corner
lon_c	Longitude: center
lon_tl	Longitude: top left corner
lon_tr	Longitude: top right corner
mapId	Map ID

Parameter	Description
name	Layer name
points	List of points on the layer
zoomDef	Default scale
zoomMax	Minimum scale
zoomMin	Maximum scale
zoomStep	Scale
<point> group parameters	
id	Object type and ID in the format TYPE:ID
layerId	Layer ID
mapId	Map ID
angle	Object icon rotation angle
geo_angle	Viewing angle (for camera, see Configuring the camera viewing angle display on the Map)
latitude	Latitude (object coordinate) The parameter has non-zero value if: 1. An external map is used for the layer background (see Configuring the external Map server), or 2. The layer is tied to geo coordinates (see Configuring map binding to coordinate grid).
longitude	Longitude (object coordinate) The parameter has non-zero value if: 1. An external map is used for the layer background (see Configuring the external Map server), or 2. The layer is tied to geo coordinates (see Configuring map binding to coordinate grid).
x	The coordinate of the point relative to the substrate along the X axis
y	The coordinate of the point relative to the substrate along the Y axis

14.4.2 Information on one map

14.4.2.1 General request format:

GET http://IP-address:port/web2/secure/kartas/{plan}/

14.4.2.2 Request parameters:

Parameter	Is required	Description
plan	Yes	Map ID

14.4.2.3 Request example:

GET http://127.0.0.1:8085/web2/secure/kartas/2

14.4.2.4 **Response example:**

```
<karta>
  <id>2</id>
  <name>This is plan of a building</name>
</karta>
```

14.4.2.5 Response parameters:

Parameter	Description
id	Map ID
name	Map name

14.4.3 The list of layers for specific map

14.4.3.1 General request format:

GET http://IP-address:port/web2/secure/kartas/{plan}/layers

14.4.3.2 Request parameters:

Parameter	Is required	Description
plan	Yes	Map ID

14.4.3.3 Request example:

GET http://127.0.0.1:8085/web2/secure/kartas/2/layers

14.4.3.4 **Response example:**

```
<layers>
   <layer>
      <geo_h>10.0</geo_h>
      <height>578</height>
      <id>2</id>
      <lat_bl>43.5905</lat_bl>
      <lat_br>43.6875</lat_br>
```

```
      <lat_c>43.6395</lat_c>
      <lat_tl>43.5914</lat_tl>
      <lat_tr>43.6885</lat_tr>
      <lon_bl>43.4584</lon_bl>
      <lon_br>43.4599</lon_br>
      <lon_c>43.4809</lon_c>
      <lon_tl>43.5018</lon_tl>
      <lon_tr>43.5033</lon_tr>
      <mapId>2</mapId>
      <name>Layer 2</name>
      <points>
         <point>
            <id>CAM:1</id>
            <layerId>2</layerId>
            <mapId>2</mapId>
            <angle>0.0</angle>
            <geo_angle>0.0</geo_angle>
            <latitude>43.47727</latitude>
            <longitude>43.602381</longitude>
            <x>95.0</x>
            <y>329.0</y>
         </point>
      </points>
      <width>800</width>
      <zoomDef>1.0</zoomDef>
      <zoomMax>4.0</zoomMax>
      <zoomMin>0.25</zoomMin>
      <zoomStep>0.25</zoomStep>
   </layer>
</layers>
```

14.4.3.5 Response parameters

Parameter	Description
<layer> group parameters	
geo_h	Height mark (see Configuring map binding to coordinate grid)
height	Layer substrate height in pixels
width	Layer substrate width in pixels
id	Layer ID
lat_bl	Latitude: bottom left corner
lat_br	Latitude: bottom right corner
lat_c	Latitude: center
lat_tl	Latitude: top left corner

Parameter	Description
lat_tr	Latitude: top right corner
lon_bl	Longitude: bottom left corner
lon_br	Longitude: bottom right corner
lon_c	Longitude: center
lon_tl	Longitude: top left corner
lon_tr	Longitude: top right corner
mapId	Map ID
name	Layer name
points	List of points on the layer
zoomDef	Default scale
zoomMax	Minimum scale
zoomMin	Maximum scale
zoomStep	Scale
<point> group parameters	
id	Object type and ID in the format TYPE:ID
layerId	Layer ID
mapId	Map ID
angle	Object icon rotation angle
geo_angle	Viewing angle (for camera, see Configuring the camera viewing angle display on the Map)
latitude	Latitude (point coordinate) The parameter has non-zero value if: 1. An external map is used for the layer background (see Configuring the external Map server), or 2. The layer is tied to geo coordinates (see Configuring map binding to coordinate grid).
longitude	Longitude (point coordinate) The parameter has non-zero value if:

Parameter	Description
	1. An external map is used for the layer background (see Configuring the external Map server), or 2. The layer is tied to geo coordinates (see Configuring map binding to coordinate grid).
x	The coordinate of the point relative to the substrate along the X axis
y	The coordinate of the point relative to the substrate along the Y axis

14.4.4 Information on a specific layer

14.4.4.1 General request format:

GET http://IP-address:port/web2/secure/kartas/{plan}/layers/{base}/

14.4.4.2 Request parameters:

Parameter	Is required	Description
plan	Yes	Map ID
base	Yes	Layer ID

14.4.4.3 Request example:

GET http://127.0.0.1:8085/web2/secure/kartas/2/layers/2/

14.4.4.4 **Response example:**

```
<layer>
    <height>1000</height>
    <id>2</id>
    <mapId>2</mapId>
    <name>Base layer for plan</name>
    <width>1000</width>
    <zoomDef>1.0</zoomDef>
    <zoomMax>4.0</zoomMax>
    <zoomMin>0.25</zoomMin>
    <zoomStep>0.25</zoomStep>
</layer>
```

14.4.4.5 Response parameters:

Parameter	Description
height	Layer substrate height in pixels
width	Layer substrate width in pixels

Parameter	Description
id	Layer ID
mapId	Map ID
name	Layer name
zoomDef	Default scale
zoomMax	Minimum scale
zoomMin	Maximum scale
zoomStep	Scale

14.4.5 Layer background

14.4.5.1 General request format:

GET http://IP-address:port/web2/secure/kartas/{plan}/layers/{base}/image.{ext}

14.4.5.2 Request parameters:

Parameter	Is required	Description
plan	Yes	Map ID
base	Yes	Layer ID
ext	Yes	File extension. Allowed values: **png** or **jpg**

14.4.5.3 Request example:

GET http://127.0.0.1:8085/web2/secure/kartas/2/layers/3/image.png

14.4.5.4 Response example:

Image in specified format comes in response.

14.4.5.5 Errors while request execution:

Error	Description
404	**JPEG** extension is specified in the request.

14.4.6 The list of point on the layer

14.4.6.1 General request format:

GET http://IP-address:port/web2/secure/kartas/{plan}/layers/{base}/points/

14.4.6.2 Request parameters:

Parameter	Is required	Description
plan	No	Map ID
base	No	Layer ID

14.4.6.3 **Request example:**

GET http://127.0.0.1:8085/web2/secure/kartas/2/layers/2/points/

14.4.6.4 **Response example:**

> **Note.**
>
> If the display type of the object on the map is different from **Image**, then an empty response is received. See also Attaching objects to the layers of interactive map

```
<points>
  <point>
    <id>CAM:1</id>
    <layerId>2</layerId>
    <mapId>2</mapId>
    <angle>0.0</angle>
    <geo_angle>0.0</geo_angle>
    <latitude>43.47727</latitude>
    <longitude>43.602381</longitude>
    <x>95.0</x>
    <y>329.0</y>
  </point>
</points>
```

14.4.6.5 Response parameters:

Parameter	Description
id	Object ID in the format "TYPE:ID", for example "CAM:1"
layerId	Layer ID
mapId	Map ID
angle	Object icon rotation angle
geo_angle	Viewing angle (for camera, see Configuring the camera viewing angle display on the Map)
latitude	Latitude (point coordinate). The parameter has non-zero value if: 1. An external map is used for the layer background (see Configuring the external Map server), or 2. The layer is tied to geo coordinates (see Configuring map binding to coordinate grid).

Parameter	Description
longitude	Longitude (point coordinate). The parameter has non-zero value if: 1. An external map is used for the layer background (see Configuring the external Map server), or 2. The layer is tied to geo coordinates (see Configuring map binding to coordinate grid).
x	X coordinate of the upper left corner of the object icon
y	Y coordinate of the upper left corner of the object icon

The coordinate plane is attached to the layer as follows:

I.e. x and y cannot be negative, but can be fractional.

14.4.7 Information on a specific point on the layer

14.4.7.1 General request format:

GET http://IP-address:port/web2/secure/kartas/{plan}/layers/{base}/points/{CAM:id}

14.4.7.2 Request parameters:

Parameter	Is required	Description
plan	No	Map ID
base	No	Layer ID
CAM:id	Yes	Object ID in the format "TYPE:ID", for example "CAM:1"

14.4.7.3 **Request example:**

GET http://127.0.0.1:8085/web2/secure/kartas/plan/layers/base/points/CAM:2

14.4.7.4 Response **example:**

```
<point>
    <id>CAM:2</id>
```

```
    <layerId>base</layerId>
    <mapId>plan</mapId>
    <x>200.0</x>
    <y>200.0</y>
</point>
```

14.4.7.5 Response parameters:

Parameter	Description
id	Object ID in the format "TYPE:ID", for example "CAM:1"
layerId	Layer ID
mapId	Map ID
x	X coordinate of the upper left corner of the object icon
y	Y coordinate of the upper left corner of the object icon

14.5 Object classes

14.5.1 The list of object classes on the server

14.5.1.1 General request format:

GET http://IP-address:port/web2/secure/objectClasses

14.5.1.2 **Request example:**

GET http://127.0.0.1:8085/web2/secure/objectClasses

14.5.1.3 Response **example:**

```
<objectClasses>
    <objectClass>
        <id>GRELE</id>
    </objectClass>
    <objectClass>
        <id>USERS</id>
    </objectClass>
    <objectClass>
        <id>CAM</id>
    </objectClass>
    <objectClass>
        <id>RIGHTS</id>
    </objectClass>
    <objectClass>
        <id>GRAY</id>
    </objectClass>
</objectClasses>
```

14.5.1.4 Response parameters:

Parameter	Description
id	Object class ID

14.5.2 Specific object class

14.5.2.1 General request format:

GET http://IP-address:port/web2/secure/objectClasses/{objectClass}/

14.5.2.2 Request parameters:

Parameter	Is required	Description
objectClass	Yes	Object class ID

14.5.2.3 **Request example:**

GET http://127.0.0.1:8085/web2/secure/objectClasses/GRELE/

14.5.2.4 Response **example:**

```
<objectClass>
    <id>GRELE</id>
</objectClass>
```

14.5.2.5 Response parameters:

Parameter	Description
id	Object class ID

14.5.3 The list of states for a specific object class

14.5.3.1 General request format:

GET http://IP-address:port/web2/secure/objectClasses/{objectClass}/states/

14.5.3.2 Request parameters:

Parameter	Is required	Description
objectClass	Yes	Object class ID

14.5.3.3 **Request example:**

GET http://127.0.0.1:8085/web2/secure/objectClasses/GRELE/states/

14.5.3.4 Response **example:**

```
<states>
    <state>
        <id>off</id>
    </state>
    <state>
        <id>on</id>
    </state>
    <state>
        <id>disabled</id>
    </state>
</states>
```

14.5.3.5 Response parameters:

Parameter	Description
id	The ID of all possible states of the object class

14.5.4 Information on a specific state

14.5.4.1 General request format:

GET http://IP-address:port/web2/secure/objectClasses/{ObjectClass}/states/{State}/

14.5.4.2 Request parameters:

Parameter	Is required	Description
ObjectClass	Yes	Object class ID
State	Yes	Object class state ID

14.5.4.3 **Request example:**

GET http://127.0.0.1:8085/web2/secure/objectClasses/GRELE/states/off/

14.5.4.4 Response **example:**

```
<state>
    <id>off</id>
</state>
```

14.5.4.5 Response parameters:

Parameter	Description
id	Object class state ID

14.5.5 Getting the icon for a specific state

14.5.5.1 General request format:

GET http://IP-address:port/web2/secure/objectClasses/{ObjectClass}/states/{State}/image.png

14.5.5.2 Request parameters:

Parameter	Is required	Description
objectClass	Yes	Object class ID
State	Yes	Object class state ID

14.5.5.3 Request example:

GET http://127.0.0.1:8085/web2/secure/objectClasses/GRELE/states/off/image.png

14.5.5.4 Response example:

The answer will be a png image.

14.5.6 The list of events for a specific object class

14.5.6.1 General request format:

GET http://IP-address:port/web2/secure/objectClasses/{ObjectClass}/events/

14.5.6.2 Request parameters:

Parameter	Is required	Description
ObjectClass	Yes	Object class ID

14.5.6.3 Request example:

GET http://127.0.0.1:8085/web2/secure/objectClasses/GRELE/events/

14.5.6.4 Response example:

```
<events>
    <event>
        <id>23</id>
        <sid>grele.disable</sid>
        <description>Disable rele</description>
    </event>
    <event>
        <id>24</id>
        <sid>grele.enable</sid>
        <description>Enable rele</description>
    </event>
</events>
```

14.5.6.5 Response parameters:

Parameter	Description
id	Object ID
sid	Event command
description	Event description

14.6 Objects

14.6.1 Getting list of all server objects

14.6.1.1 General request format:

GET http://IP-address:port/web2/secure/configuration?pageItems={pageItems}&page={page}

14.6.1.2 Request parameters:

Parameter	Is required	Description
pageItems	No	Sets the number of objects displayed on the page. The value must be greater than 0, by default page=1000. The **page** parameter only applies when the **pageItems** parameter is specified
page	No	Sets the page number displayed as a result of the request. The value must be greater than 0, by default pageItems=1

> **Attention!**
>
> If there are many objects in the system (>1000) they are to be displayed by pages.

Processing of all objects is performed page by pare until an empty array is received.

14.6.1.3 Request example:

GET http://127.0.0.1:8085/web2/secure/configuration

14.6.1.4 Response example:

The request returns the list of the following objects with states:

- cameras added to the Web-server with IDs of linked microphones, dynamics, PTZ devices, presets, as well as monitors and displays to which the camera added (see also Selecting and configuring cameras for the Web-server module);
- displays and monitors on which the cameras added to the Web-server are displayed;
- cameras added to maps selected for Web Server 2.0 are returned – see Selecting maps;
- sensors;
- relays;
- macros;
- RTSP servers with ports used, cameras added;
- list of zones and regions.

JSON:

```
[
    {
        "id": "1",
        "name": "Area 1",
        "regions": [
            {
                "id": "1.1",
                "zoneId": "1",
                "name": "Region 1.1",
                "zoneDescription": "Zone description"
            }
        ]
    },
    {
        "type": "DISPLAY",
        "id": "DISPLAY:1",
        "extId": "1",
        "name": "Display 1",
        "displayId": "1",
        "state": {
            "id": "normal",
            "type": "NORMAL",
            "fullState": null
        }
    },
    {
        "type": "MACRO",
        "id": "MACRO:2",
        "extId": "2",
        "name": "Macro 2",
        "state": {
            "id": "normal",
            "type": "NORMAL",
            "fullState": null
        }
    },
    {
        "type": "STREAMING_SERVER",
        "id": "STREAMING_SERVER:1",
        "extId": "1",
        "name": "RTSP Server 1",
        "state": {
            "id": "normal",
            "type": "NORMAL",
            "fullState": null
        },
        "port": "554",
        "cams": "1;;;;;;"
    },
    {
        "type": "CAM",
        "id": "CAM:1",
        "extId": "1",
        "name": "Camera 1",
        "displayId": "1",
        "monitorId": "1",
```

```
        "state": {
            "id": "connected_recording",
            "type": "NORMAL",
            "fullState": "DISARMED|RECORDER_ON|RECORDING"
        },
        "presets": []
    },
    {
        "type": "SLAVE",
        "id": "SLAVE:R-GYZYEV",
        "extId": "R-GYZYEV",
        "name": "Computer DESKTOP-JHRURJJ",
        "state": {
            "id": "connected",
            "type": "NORMAL",
            "fullState": null
        }
    },
    {
        "type": "MONITOR",
        "id": "MONITOR:1",
        "extId": "1",
        "name": "Monitor 1",
        "displayId": "1",
        "monitorId": "1",
        "state": {
            "id": "normal",
            "type": "NORMAL",
            "fullState": null
        },
        "camList": [
            "1"
        ]
    }
]
```


XML:


```
<baseObjects>
   <CAM>
      <displayId>1</displayId>
      <extId>9</extId>
      <geo_angle>0.0</geo_angle>
      <id>CAM:9</id>
      <latitide>-98.0533</latitide>
      <longitude>56.4089</longitude>
      <monitorId>1;2</monitorId>
      <name>Camera 9</name>
      <regionId />
      <state>
         <fullState>DISARMED</fullState>
         <id>connected</id>
         <type>NORMAL</type>
      </state>
      <type>CAM</type>
      <additionalInfo />
```

```
      <micId />
      <presets />
      <speakerId />
      <telemetryId />
   </CAM>
   <baseObject>
      <displayId>1</displayId>
      <extId>1</extId>
      <id>DISPLAY:1</id>
      <name>Display 1</name>
      <state>
         <id>normal</id>
         <type>NORMAL</type>
      </state>
      <type>DISPLAY</type>
   </baseObject>
   <CAM>
      <displayId>1</displayId>
      <extId>5</extId>
      <geo_angle>0.0</geo_angle>
      <id>CAM:5</id>
      <latitide>-97.6674</latitide>
      <longitude>56.3588</longitude>
      <monitorId>1</monitorId>
      <name>pos</name>
      <regionId />
      <state>
         <fullState>DISARMED</fullState>
         <id>connected</id>
         <type>NORMAL</type>
      </state>
      <type>CAM</type>
      <additionalInfo />
      <micId />
      <presets />
      <speakerId />
      <telemetryId />
   </CAM>
   <STREAMING_SERVER>
      <cams>1;2;3;4;5;6;9;;;;;;;;;;</cams>
      <extId>1</extId>
      <id>STREAMING_SERVER:1</id>
      <name>RTSP Server 1</name>
      <state>
         <id>normal</id>
         <type>NORMAL</type>
      </state>
      <type>STREAMING_SERVER</type>
      <port>5543</port>
   </STREAMING_SERVER>
   <CAM>
      <displayId>1</displayId>
      <extId>1</extId>
      <geo_angle>0.0</geo_angle>
      <id>CAM:1</id>
      <latitide>-98.3293</latitide>
      <longitude>56.741</longitude>
```

```
      <monitorId>1;2</monitorId>
      <name>Camera 1</name>
      <regionDescription />
      <regionId>1.1</regionId>
      <regionName>Region 1.1</regionName>
      <state>
         <fullState>DISARMED</fullState>
         <id>connected</id>
         <type>NORMAL</type>
      </state>
      <type>CAM</type>
      <zoneId>1</zoneId>
      <zoneName>Area 1</zoneName>
      <additionalInfo />
      <micId />
      <presets />
      <speakerId />
      <telemetryId />
   </CAM>
   <SLAVE>
      <extId>R-GYZYEV</extId>
      <id>SLAVE:R-GYZYEV</id>
      <name>LOCALHOST</name>
      <state>
         <id>connected</id>
         <type>NORMAL</type>
      </state>
      <type>SLAVE</type>
   </SLAVE>
   <baseObject>
      <displayId>1</displayId>
      <extId>2</extId>
      <id>MONITOR:2</id>
      <monitorId>2</monitorId>
      <name>Monitor 2</name>
      <state>
         <id>normal</id>
         <type>NORMAL</type>
      </state>
      <type>MONITOR</type>
   </baseObject>
   <baseObject>
      <displayId>1</displayId>
      <extId>1</extId>
      <id>MONITOR:1</id>
      <monitorId>1</monitorId>
      <name>Monitor 1</name>
      <state>
         <id>normal</id>
         <type>NORMAL</type>
      </state>
      <type>MONITOR</type>
   </baseObject>
</baseObjects>
```

14.6.1.5 Response parameters:

Parameter	Description
General parameters	
extId	Object ID
id	Object type and ID in the format TYPE:ID
name	Object name
state	Object state. The <id>, <type> and <fullState> description see in State of a specific object
type	Object type
Specific parameters	
cams	List of cameras added to the RTSP Server with semicolon as a separator
port	Port used by RTSP Server
regionDescription	Region description
regionId	ID of the region to which the object is added
regionName	Region name
zoneId	ID of the zone to which the object is added
zoneName	Zone name
latitide	Latitude (object coordinate). The parameter has non-zero value if: 1. An external map is used for the layer background (see Configuring the external Map server), or 2. The layer is tied to geo coordinates (see Configuring map binding to coordinate grid)
longitude	Longitude (object coordinate). The parameter has non-zero value if: 1. An external map is used for the layer background (see Configuring the external Map server), or 2. The layer is tied to geo coordinates (see Configuring map binding to coordinate grid)
monitorId	For camera: ID of the monitor to which the camera is added. If there are several IDs, they are separated by a semicolon ";"
monitorName	For monitor: name of the monitor
geo_angle	For camera: viewing angle (for camera, see Configuring the camera viewing angle display on the Map)
additionalInfo	For camera: the **Additional info** field value
micId	For camera: the ID of the associated microphone
presets	For camera: preset list
speakerId	For camera: the ID of the associated speaker
telemetryId	For camera: the ID of the PTZ control panel

Parameter	Description
displayId	For interface objects: display ID. If there are several IDs, they are separated by a semicolon ";"
displayName	For display: display name
camList	For monitor: IDs of cameras added to the monitor separated by semicolon ";"

14.6.2 Information on a specific object

14.6.2.1 General request format:

GET http://IP-address:port/web2/secure/configuration/{objectClass}:{id}/

14.6.2.2 Request parameters:

Parameter	Is required	Description
ObjectClass	Yes	Object class ID
id	Yes	Object ID

14.6.2.3 **Request example**:

GET http://127.0.0.1:8085/web2/secure/configuration/GRAY:2/

14.6.2.4 **Response example**:

```
<GRAY>
    <id>GRAY:2</id>
    <name>Gray 2</name>
    <state>
        <id>alarmed</id>
    </state>
</GRAY>
```

14.6.2.5 Response parameters:

Parameter	Description
id	Object ID in the format "TYPE:ID", for example "CAM:1"
name	Object name in *Intellect*
state id	The current state of the object

14.6.3 State of a specific object

14.6.3.1 General request format:

GET http://IP-address:port/web2/secure/configuration/{objectClass}:{id}/state/

14.6.3.2 Request parameters:

Parameter	Is required	Description
objectClass	Yes	Object class ID
id	Yes	Object ID

14.6.3.3 **Request example**:

GET http://127.0.0.1:8085/web2/secure/configuration/GRAY:2/state/

14.6.3.4 Response **example**:

```
<GRAY>
    <id>GRAY:2</id>
    <name>Gray 2</name>
    <state>
        <id>alarmed</id>
    </state>
</GRAY>
```

14.6.3.5 Response parameters:

Parameter	Description
fullState	Full object state as stored in the database
id	Object state in terms of HTTP API
type	Object state in terms of HTTP API

Possible values of fullState parameter for a sensor are as follows:

Sensor state	**fullState in web request**	dbo.state
Armed + Closed	ON,ARMED	ON|ARMED
Armed + Closed+ Alarm	ON,ALARMED	ON|ALARMED
Armed + Closed+ Alarm confirmed	ON,CONFIRMED	ON|CONFIRMED
Armed + Closed+ Connection lost	ON,DETACHED_DISARM	ON|DETACHED_DISARM
Disarmed + Closed	ON,DISARMED	ON|DISARMED
Disarmed + Closed+ Alarm	ON,ALARMED	ON|ALARMED
Disarmed + Closed+ Alarm confirmed	ON,CONFIRMED	ON|CONFIRMED
Disarmed + Closed+ Connection lost	ON,DETACHED_DISARM	ON|DETACHED_DISARM
Armed + Opened	ARMED,OFF	ARMED|OFF
Armed + Opened+ Alarm	OFF,ALARMED	OFF|ALARMED

Sensor state	**fullState in web request**	dbo.state
Armed + Opened+ Alarm confirmed	OFF,CONFIRMED	OFF|CONFIRMED
Armed + Opened+ Connection lost	DETACHED_DISARM,OFF	DETACHED_DISARM|OFF
Disarmed + Opened	DISARMED,OFF	DISARMED|OFF
Disarmed + Opened+ Alarm	OFF,ALARMED	OFF|ALARMED
Disarmed + Opened+ Alarm confirmed	OFF,CONFIRMED	OFF|CONFIRMED
Disarmed + Opened+ Connection lost	DETACHED_DISARM,OFF	DETACHED_DISARM|OFF

14.6.4 The list of available actions with the object in a specific state

The list of actions is requested not by the object class, but is taken in the context of a specific object as various user rights are possible for the objects of the same class. Information on how to use this list is given in Sending commands to server section.

14.6.4.1 General request format:

GET http://IP-address:port/web2/secure/configuration/{objectClass}:{id}/state/actions/

14.6.4.2 Request parameters:

Parameter	Is required	Description
ObjectClass	Yes	Object class ID
id	Yes	Object ID

14.6.4.3 **Request example**:

GET http://127.0.0.1:8085/web2/secure/configuration/GRAY:2/state/actions/

14.6.4.4 Response **example**:

XML

```
<actions>
    <action>
        <description>Disarm</description>
        <hidden>false</hidden>
        <id>DISARM</id>
    </action>
    <action>
        <description>Arm</description>
        <hidden>false</hidden>
        <id>ARM</id>
    </action>
    <action>
        <description>Classify alarm</description>
        <hidden>false</hidden>
        <id>CONFIRM</id>
    </action>
</actions>
```

If the object state considers no actions, then xml is:

```
<actions/>
```

14.6.4.5 JSON

```
[
    {
        "id": "DISARM",
        "description": "Disarm",
        "hidden": false,
        "react": {
            "sourceType": "MACRO",
            "sourceId": "DISARM",
            "action": "RUN",
            "params": {},
            "name": "React",
            "version": 1
        }
    },
    {
        "id": "ARM",
        "description": "Arm",
        "hidden": false,
        "react": {
            "sourceType": "MACRO",
            "sourceId": "ARM",
            "action": "RUN",
            "params": {},
            "name": "React",
            "version": 1
        }
    },
    {
        "id": "CONFIRM",
```

```
        "description": "Classify alarm",
        "hidden": false,
        "react": {
            "sourceType": "MACRO",
            "sourceId": "CONFIRM",
            "action": "RUN",
            "params": {},
            "name": "React",
            "version": 1
        }
    }
]
```

14.6.4.6 Response parameters:

Parameter	Description
description	User-friendly name of the reaction
hidden	true – the reaction is not displayed in the interface (on the map, in macros, etc.) false – the reaction is displayed in the interface
id	System name of the reaction
"react" block in the JSON response	
sourceType	Type of object that can run action
sourceId	System name of the reaction
action	Reaction of the sourceType
params	Available parameters of the reaction
name	Action type: Event or React
version	Version

14.6.5 Getting list of all zones and regions

14.6.5.1 General request format:

GET http://IP-address:port/web2/secure/configuration/zones

14.6.5.2 **Request example**:

GET http://127.0.0.1:8085/web2/secure/configuration/zones

14.6.5.3 **Response example**:

JSON:

```
[
    {
```

```
        "id": "1",
        "name": "Zone 1",
        "regions": [
            {
                "id": "1.1",
                "zoneId": "1",
                "name":"Region 1.1",
                "zoneDescription": "Zone description"
            }
        ]
    },
    {
        "id": "2",
        "name": "Zone 2",
        "regions": [
            {
                "id": "2.1",
                "zoneId": "2",
                "name": "Region 2.1",
                "zoneDescription": "Zone description"
            }
        ]
    }
]
```


XML:


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<zones>
    <zone>
        <id>1</id>
        <name>Zone 1</name>
        <regions>
            <id>1.1</id>
            <name>Region 1.1</name>
            <zoneDescription>Zone description</zoneDescription>
            <zoneId>1</zoneId>
        </regions>
    </zone>
    <zone>
        <id>2</id>
        <name>Zone 2</name>
        <regions>
            <id>2.1</id>
            <name>Region 2.1</name>
            <zoneDescription>Zone description</zoneDescription>
            <zoneId>2</zoneId>
        </regions>
    </zone>
</zones>
```

14.6.5.4 Response parameters:

Parameter	Description
id	ID of the zone/region
name	Name of the zone/region in *Intellect*
zoneDescription	Zone description

14.7 Getting events

Getting events from the cameras that changed their state during the request execution (~30-60 seconds).

14.7.1 General request format:

http://IP-address:port/web2/secure/feed/

14.7.2 **Request example**:

http://127.0.0.1:8085/web2/secure/feed/

14.7.3 Response **example**:

```
<message>
    <action>update</action>
    <objectId>CAM:1</objectId>
    <state>disconnected</state>
</message>

<message>
    <action>state</action>
    <objectId>CAM:1</objectId>
    <x>10.0</x>
    <y>123.9</y>
</message>

<message>
    <action>state</action>
    <objectId>CAM:1</objectId>
    <state>connected</state>
    <x>300.8</x>
    <y>670</y>
</message>

<message>
    <action>state</action>
    <objectId>CAM:1</objectId>
    <x>100</x>
    <y>100</y>
</message>

<message>
    <action>ping</action>
</message>
```

14.7.4 Response parameters:

Parameter	Description
action	Type of event. Possible values: create, delete, update
objectId	id of the object that is the source of event (always with update, delete, create)
state	id of a new object state (always with create. If the state has not changed, then there is no update state).
x, y	New coordinates, if changed
ts	The time and date of the object state change

14.7.5 Getting events of video subsystem in blocks

14.7.5.1 General request format:

GET http://IP-address:port/web2/secure/events/

14.7.5.2 Request parameters:

Parameter	Description
from	The oldest date of message search period. Example: 2012-12-27T15%3A19%3A16.000%2B03%3A00
to	The latest date of message search period. Example: 2012-12-27T15%3A19%3A16.000%2B03%3A00
count	Maximum number of messages in reply in the range [1, 200]. Default is 20. Server can return more messages if there are few messages in the database
objectId	Object class and object ID, which are separated by a colon. Examples: CAM:1, GRAY:5, PEOPLE_COUNTER:1 etc. Events can be received from several objects separated by commas. Example: objectId=CAM:1,PEOPLE_COUNTER:1 − the response will contain events for camera 1 and counter 1. If the object ID is not specified in the parameter, then the events by all objects of the specified class will be returned. Example: objectId=CAM − the response will contain events by all cameras
action	Event type. If the parameter is specified, then the response will contain only events of the specified type. Examples: • REC − start recording • REC_STOP − stop recording • ARM − arming the camera • DISARM − disarming the camera • disconnected − loss of connection with the camera

14.7.5.3 Request example:

GET http://127.0.0.1:8085/web2/secure/events?
from=2021-05-26T14%3A30%3A30.000%2B03%3A00&to=2021-05-26T15%3A40%3A30.000%2B03%3A00&count=5&objectId=MACRO:7

14.7.5.4 Response example:

XML:

```
<events>
    <event>
        <description>Action executed</description>
        <id>{3AED63A0-19BE-EB11-9020-B42E99FDB342}</id>
        <objectId>MACRO:7</objectId>
        <addInfo>event1</addInfo>
        <params0>User 1</params0>
        <params1>1</params1>
        <params2/>
        <params3/>
        <ts>2021-05-26T14:58:05+03:00</ts>
        <type>Empty</type>
    </event>
    <event>
        <description>Action executed</description>
        <id>{26840B9A-19BE-EB11-9020-B42E99FDB342}</id>
        <objectId>MACRO:7</objectId>
        <addInfo>event2</addInfo>
           params0>User 1</params0>
        <params1>1</params1>
        <params2/>
        <params3/>
        <ts>2021-05-26T14:58:04+03:00</ts>
        <type>Empty</type>
    </event>
</events>
```

JSON:

```
[
    {
        "id": "{3AED63A0-19BE-EB11-9020-B42E99FDB342}",
        "objectId": "MACRO:7",
        "ts": "2021-05-26T14:58:05.000+03:00",
        "description": "Action executed",
        "addInfo": "event1",
        "type": "Empty",
        "params2": "",
        "params3": "",
        "params1": "1",
        "params0": "User 1"
    },
    {
        "id": "{26840B9A-19BE-EB11-9020-B42E99FDB342}",
        "objectId": "MACRO:7",
        "ts": "2021-05-26T14:58:04.000+03:00",
        "description": "Action executed",
        "addInfo": "event2",
        "type": "Empty",
        "params2": "",
        "params3": "",
        "params1": "1",
        "params0": "User 1"
    }
]
```

Return codes:

200 – OK
400 – invalid parameter (e.g. date format)
500 – error
503 – error of core connection
504 – time-out (core failed to return data within 2000 milliseconds)

14.8 Sending commands to server

14.8.1 General request format:

PUT http://IP-address:port/web2/secure/configuration/{objectClass}:{id}/state/actions/{CMD}/execute

14.8.2 Request example:

Parameter	Is required	Description
objectClass	Yes	Object class ID
id	Yes	Object ID
CMD	Yes	Command *Attention! The command name must be in uppercase.*

14.8.3 Response example:

PUT http://127.0.0.1:8085/web2/secure/configuration/GRAY:2/state/actions/DISARM/execute

14.9 Macros

Macros - predefined sequence of responses to certain events. Macros are created on the server and have IDs and names. They are similar to actions with objects, but are not attached to the object.

14.9.1 Getting parameters of macros

14.9.1.1 General request format:

GET http://IP-address:port/web2/secure/actions/

14.9.1.2 **Request parameters:**

Parameter	Required	Description
id	Yes	Mac

Parameter	Required	Description
		ro ID

14.9.1.3 **Request example**:

GET http://127.0.0.1:8085/web2/secure/actions/

14.9.1.4 Response **example**:

```
<action>
    <description>Start recording for all cameras</description>
    <id>2</id>
</action>
```

14.9.1.5 Response parameters:

Parameter	Description
description	Macro name
id	Macro ID

14.9.2 Getting the list of macros

14.9.2.1 General request format:

GET http://IP-address:port/web2/secure/actions/{id}/

14.9.2.2 Request parameters:

Parameter	Is required	Description
id	Yes	Macro ID

14.9.2.3 **Request example**:

GET http://127.0.0.1:8085/web2/secure/actions/2/

14.9.2.4 Response **example**:

```
<action>
```

```
    <description>Start recording by all cameras</description>
    <id>2</id>
</action>
```

14.9.2.5 Response parameters:

Parameter	Description
description	Macro name
id	Macro ID

14.9.3 Macros execution on server request

14.9.3.1 General request format:

PUT http://IP-address:port/web2/secure/configuration/{id}/state/actions/RUN/execute

14.9.3.2 **Request parameters**:

Parameter	Is required	Description
id	Да	Macro type and ID in the format TYPE:ID (for example MACRO:1)

14.9.3.3 **Request example**:

PUT http://127.0.0.1:8085/web2/secure/configuration/MACRO:1/state/actions/RUN/execute

14.10 Video

14.10.1 Thumbnails request

14.10.1.1 General request format:

14.10.1.1.1 First way

GET http://IP-address:port/web2/secure/video/image.jpg?cam.id={cam.id}&width={width}&height={height}&version={version}&login={login}&password={password}

14.10.1.2 Request parameters:

Parameter	Is required	Description
cam.id	Yes	Camera ID
width	No	Value can be in [64, 1600] range. Server automatically rounds width to larger value divisible by 4 Size of returned image is taken from video stream if width parameter are not set.

Parameter	Is required	Description
height	No	Value can be in [30, 1200] range. Size of returned image is taken from video stream if height parameter are not set.
version	No	See Product version
login	No	*Intellect* username, if set
password	No	*Intellect* user password, if set

14.10.1.2.1 Second way

GET http://IP-address:port/web2/secure/video/action.do?version={version}&command=frame.video&video_in={video_in}&imageWidth={imageWidth}&imageHeight={imageHeight}&login={login}&password={password}

14.10.1.3 Request parameters:

Parameter	Is required	Description
version	No	See Product version
video_in	Yes	Camera ID in the format "TYPE:ID", for example "CAM:1"
imageWidth	No	Value can be in [64, 1600] range. Server automatically rounds width to larger value divisible by 4 Size of returned image is taken from video stream if width parameter are not set.
imageHeight	No	Value can be in [30, 1200] range. Size of returned image is taken from video stream if height parameter are not set.
login	No	*Intellect* username, if set
password	No	*Intellect* user password, if set

14.10.1.4 **Request example**:

14.10.1.4.1 First way

GET http://127.0.0.1:8085/web2/secure/video/image.jpg?cam.id=5&width=85&version=4.7.8.0&login=USER&password=PASS

14.10.1.4.2 Second way

GET http://127.0.0.1:8085/web2/secure/video/action.do?version=4.9.0.0&command=frame.video&video_in=CAM:1&imageWidth=400

14.10.1.5 Response **example**:

Jpg image of approximately requested size or error code and zero length body (i.e. only headings) will be received in reply.

In case of error the http error code is returned:
404 – camera disabled or not in use (disabled);
403 – invalid password;
426 – old client version;

429 – too many requests;
444 – camera signal is lost or camera disabled (coaxial conductor disconnected from card);
503 – archive error.

14.10.2 Configuration request

14.10.2.1 General request format:

GET http://IP-address:port/web2/secure/video/config.properties?version={version}&login={login}&password={password}

14.10.2.2 Request parameters:

Parameter	Is required	Description
version	Yes	See Product version
login	No	*Intellect* username, if set
password	No	*Intellect* user password, if set

14.10.2.3 **Request example**:

GET http://127.0.0.1:8085/web2/secure/video/config.properties?version=4.7.8.0&login=USER&password=PASS

14.10.2.4 Response **example**:

Config.properties text file.

If a password is set but not specified in the request:

```
password.enabled=true
login.enabled=true
password.invalid=true#
```

If the password is correct or access is allowed without the password, then the server sends the following configuration:

```
password.enabled=true
login.enabled=true
password.invalid=false
cam.0.id=2
cam.0.name=Face
cam.0.rights=11
cam.1.id=3
cam.1.name=Camera 3
cam.1.rights=11
cam.2.id=5
cam.2.name=Camera 5
cam.2.rights=11
cam.2.telemetry_id=1.1
cam.count=3#
```

14.10.2.5 **Response** parameters:

Parameter	Description
password.enabled	false - no password required true - password required
login.enabled	false - no login required true - login required
password.invalid	false - specified password is correct true - specified password is not correct
cam.count	Total count of cameras in the configuration (id starts at 0).
cam.N.id	Camera ID
cam.N.name	Camera name

cam.N.rights	Rights (they are checked on the server; available on the client in order not to show the user odd options). The parameter is represented by flags. If the flag is set, then the interface element is to be shown; if not – hidden.
	static final int RIGHT_VIEW = 0x1; // live video is available (always 1)
	static final int RIGHT_CONTROL = 0x2; // control (telemetry, arming and disarming)
	static final int RIGHT_CONFIG = 0x4; // reserved
	static final int RIGHT_HISTORY = 0x8; // access to archive

cam.N.telemetry_id	Telemetry ID (there can be no value if there is no telemetry – then hide telemetry control elements).

14.10.3 Video query

14.10.3.1 General request format:

GET http://IP-address:port/web2/secure/video/action.do?version={version}&sessionid={sessionid}&video_in={video_in}&normalize={normalize}&imageWidth={imageWidth}&imageHeight={imageHeight}&fps={fps}&login={login}&password={password}

14.10.3.2 Request parameters:

Parameter	Is required	Description
version	Yes	See Product version. If version 4.10.0.0 is specified in request, then the *stream obtained* is in *MJPEG* format without XML inserts – it can be displayed on a web page using the IMG tag in Chrome and FireFox browsers. This feature is implemented for both live and archive video. *Note. Not more than 6 video streams can be received simultaneously.*
video_in	Yes	Camera ID in the format "TYPE:ID", for example "CAM:1"
sessionid	No	Session ID
imageWidth	No	Frame width. Value can be in [64, 1600] range. Server automatically rounds width to larger value divisible by 4

Parameter	Is required	Description
imageHeight	No	Frame height. Value can be in [30, 1200] range. If not, then the size of the returned image is taken from the video stream.
normalize	No	true - stretches the image if the frame comes in incorrect proportions
fps	No	Video frame rate
login	No	*Intellect* username, if set
password	No	*Intellect* user password, if set

14.10.3.3 **Request example**:

GET http://127.0.0.1:8085/web2/secure/video/action.do?version=4.10.0.0&sessionid=1234567890&video_in=CAM:1&imageWidth=200&fps=1&login=USER&password=PASS

14.10.3.4 Response **example**:

```
<html>
 <head/>
 <body>
  <img src="path to mjpeg"/>
 </body>
</html>
```

14.10.4 Curl video request. Main stream format

14.10.4.1 General request format:

curl -v "http://IP-address:port/web2/secure/video/action.do?fps={fps}&imageHeight={imageHeight}&login={login}&normalize={normalize}&password={password}&sessionid={sessionid}&version={version}&video_in={video_in}" --output ~/{output file}.bin

14.10.4.2 Request parameters:

Parameter	Is required	Description
version	Yes	See Product version. If version 4.10.0.0 is specified in request, then the *stream obtained* is in *MJPEG* format without XML inserts – it can be displayed on a web page using the IMG tag in Chrome and FireFox browsers. This feature is implemented for both live and archive video. *Note. Not more than 6 video streams can be received simultaneously.*
video_in	Yes	Camera ID in the format "TYPE:ID", for example "CAM:1"
sessionid	No	Session ID

Parameter	Is required	Description
imageWidth	No	Frame width. Value can be in [64, 1600] range. Server automatically rounds width to larger value divisible by 4. If not, then the size of the returned image is taken from the video stream.
imageHeight	No	Frame height. Value can be in [30, 1200] range. If not, then the size of the returned image is taken from the video stream.
normalize	No	true - stretches the image if the frame comes in incorrect proportions
fps	No	Video frame rate
login	No	*Intellect* username, if set
password	No	*Intellect* user password, if set
output file name	Yes	The name of the file to which the requested video will be output

14.10.4.3 **Request example**:

curl -v "http://127.0.0.1:8085/web2/secure/video/action.do?fps=1&imageHeight=360&login=1&normalize=true&password=1&sessionid=A4D98DDE-A535-49E4-9FB5-FAD441CBBA43&version=4.10.0.0&video_in=CAM:1" --output ~/output.bin

14.10.4.4 Response **example**:

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0* Connected to
172.19.2.6 (172.19.2.6) port 8085 (#0)
* Server auth using Basic with user '1'
> GET /web2/secure/video/action.do?
fps=1&imageHeight=360&login=1&normalize=true&password=1&sessionid=A4D98DDE-A535-49E4-9FB5-
FAD441CBBA43&version=4.10.0.0&video_in=CAM:1 HTTP/1.1
> Host: 172.19.2.6:8085
> Authorization: Basic MTox
> User-Agent: curl/7.74.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Set-Cookie: JSESSIONID=ae532304-e6dc-4a19-8c4f-1e10656140f2; Path=/web2; HttpOnly
< Set-Cookie: rememberMe=deleteMe; Path=/web2; Max-Age=0; Expires=Sun, 24-Jan-2021 08:09:52 GMT
< Server: ITV-Intellect-Webserver/4.11.2.3000
< Date: Mon, 25 Jan 2021 08:09:52 GMT
< Cache-Control: no-store, no-cache, must-revalidate, max-age=0
< Connection: close
< Content-Type: multipart/x-mixed-replace;boundary=videoframe
< Pragma: no-cache
< X-CameraID: 1
< X-SessionID: A4D98DDE-A535-49E4-9FB5-FAD441CBBA43
< Closing connection 0
```

```
<
{ [32768 bytes data]
```

14.10.4.5 Response parameters:

The Content-Type header contains the following parameters:

Parameter	Description
multipart/x-mixed-replace	Indicates that each subsequent piece of content should replace the previous one
boundary	Indicates a text separator between pieces of content

14.10.4.6 Response **example**:

The output file will contain the following parameters for each piece of requested content:

```
--videoframe
Content-Type: image/jpeg
Content-Length: 93081
X-Width: 480
X-Height: 360
X-Timestamp: 0.000
X-Time: 2021-01-25T10:06:42.816+03:00
```

14.10.4.7 Response parameters:

Parameter	Description
Content-Type	Type of content
Content-Length	Piece of content size
X-Width	Image width
X-Height	Image height
X-Time	Absolute frame time
X-Timestamp	Relative frame time in seconds (relative to the beginning of the stream)

14.10.4.8 Response **example**:

If the stream ends due to the server failure, a final packet may be received:

```
--videoframe
Content-Type: image/jpeg
Content-Length: 106
<video_in>
  <sessionid>A4D98DDE-A535-49E4-9FB5-FAD441CBBA43</sessionid>
```

```
  <video_in>CAM:1</video_in>
  <newstate>closed</newstate>
  <errcode>103</errcode>
</video_in>
```

14.10.4.9 Response parameters:

Parameter	Description
sessionid	session id (same as at start)
video_in	Camera ID in the "TYPE:ID" format, for example, "CAM:1"
errcode	Error code: • 100 - no error. • 101 - too many connected users. • 102 - incorrect password (theoretically, the password can be changed at any time of work). • 103 - video not available. • 104 - old version of the client. Update the version.

14.10.5 Format of main stream

14.10.5.1 **Example request:**

```
HTTP/1.0 200 OK
Connection: close
Server: ITV-Intellect-Webserver/4.9.0.0
Cache-Control: no-store,no-cache,must-revalidate,max-age=0
Pragma: no-cache
Date: Mon, 13 Jan 2013 10:44:27 GMT
Content-Type: multipart/mixed;boundary=videoframe

--videoframe
Content-Type: text/xml
Content-Length: 138

<video_in>
  <sessionid>FC126734</sessionid>
  <video_in>CAM:5</video_in>
  <newstate>started</newstate>
  <errcode>100</errcode>
</video_in>
 --videoframe
Content-Type: image/jpeg
Content-Length: 23978
X-Width: 320
X-Height: 240
X-Time: 2013-03-15T10:51:44.314+04:00
X-Timestamp: 0.000
```

```
 <jpeg image>
--videoframe
Content-Type: image/jpeg
Content-Length: 23651
X-Width: 320
X-Height: 240
X-Time: 2013-03-15T10:51:44.314+04:00
X-Timestamp: 0.152

 <jpeg image>
```

14.10.5.2 **Response** parameters:

Parameter	Description
X-Width	Image width
X-Height	Image height
X-Time	Absolute time of frame forming
X-Timestamp	Relative frame time in seconds (relatively stream head)

14.10.5.3 **Example response:**

In case of stream end due to the fault of server, the end packet can be received:

```
--videoframe
Content-Type: text/xml
Content-Length: 106
<video_in>
  <sessionid>FC126734</sessionid>
  <video_in>CAM:5</video_in>
  <newstate>closed</newstate>
  <errcode>103</errcode>
</video_in>
```

14.10.5.4 **Response** parameters:

Parameter	Description
sessionid	Session ID (the same as at start).
video_in	Camera ID
errcode	Error code: • 100 – error absence. • 101 – too many connected users. • 102 – invalid password (theoretically, it's possible to change password at any moment).

Parameter	Description
	• 103 – unavailable video. • 104 – old client version. Update version.

14.10.6 Camera managing

14.10.6.1 General request format:

GET http://IP-address:port/web2/secure/video/action.do?version={version}&sessionid={sessionid}&cam.id={cam.id}
&target=CAM&targetid={targetid}&command={command}&login={login}&password={password}

14.10.6.2 Request parameters:

Parameter	Is required	Description
version	Yes	See Product version
cam.id	Yes	Camera ID
sessionid	No	Session ID
target	Yes	CAM—"Camera" object class ID
targetid	Yes	Matches cam.id
command	Yes	Camera control commands: • Start recording: REC • Stop recording: REC_STOP • Arming: ARM • Disarming: DISARM
manual<number>	Yes	Manual recording mode. The parameter is automatically generated for Start recording and Stop recording commands. <number> can be 0 (manual recording mode disabled) or 1 (manual recording mode enabled)
login	No	*Intellect* username, if set
password	No	*Intellect* user password, if set

14.10.6.3 Request example:

GET http://127.0.0.1:8085/web2/secure/video/action.do?
version=4.10.0.0&sessionid=29101F1&cam.id=1&target=CAM&targetid=1&command=REC&parameters=manual<1>

GET http://127.0.0.1:8085/web2/secure/video/action.do?
version=4.10.0.0&sessionid=29101F1&cam.id=1&target=CAM&targetid=1&command=REC_STOP&parameters=manual<1>

GET http://127.0.0.1:8085/web2/secure/video/action.do?
version=4.10.0.0&sessionid=29101F1&cam.id=1&target=CAM&targetid=1&command=ARM

GET http://127.0.0.1:8085/web2/secure/video/action.do?
version=4.10.0.0&sessionid=29101F1&cam.id=1&target=CAM&targetid=1&command=DISARM

PTZ control

14.10.7 Authorization by token

To receive a pre-authorized link to the camera (to receive both live or archived video), you must:

- Execute a request to get a token;
- Execute a request for the received token.

14.10.7.1 General request format to get a token:

GET http://IP-address:port/web2/secure/video/action.do?version={version}&sessionid={sessionid}&video_in={video_in}&enable_token_auth={enable_token_auth}&valid_token_hours={valid_token_hours}&login={login}&password={password}

14.10.7.2 Request parameters:

Parameter	Is required	Description
version	Yes	See Product version
video_in	Yes	Camera ID in the format "TYPE:ID", for example "CAM:1"
sessionid	No	Session ID
targetid	Yes	Matches cam.id
login	No	*Intellect* username, if set
password	No	*Intellect user password, if set*
enable_token_auth	Yes	Enable authorization by token **=1**.
valid_token_hours	No	Signature validation time (in hours). The maximum value is a week. The default value is 12 hours.

14.10.7.3 **Request example**:

```
GET http://127.0.0.1:8085/web2/secure/video/action.do?
version=4.10.0.0&sessionid=FC126734&video_in=CAM:1&enable_token_auth=1&valid_token_hours=
1&login=USER&password=PASS
```

14.10.7.4 Response **example**:

```
{
"path" : "action.do?hmac=GAqUa429sjY2E9jCTpuYeaMqReW3Y7HI"
}
```

14.10.7.5 Response parameters:

Parameter	Description
hmac	Token

14.10.7.6 General request format for the received token via **Web server 2.0**:

GET http://IP-address:port/web2/secure/video/action.do?hmac={hmac}

14.10.7.7 Request parameters:

Parameter	Is required	Description
hmac	Yes	Earlier received token

14.10.7.8 **Request example**:

GET http://127.0.0.1:8085/web2/secure/video/action.do?hmac=GAqUa429sjY2E9jCTpuYeaMqReW3Y7HI

14.10.7.9 General request format for the received token via **Web-server 1**:

> **Note**
>
> This request method is deprecated. It is recommended to make a request via the **Web server 2.0** module.

GET http://IP-address:{port}/action.do?hmac={hmac}

14.10.7.10 Request parameters:

Parameter	Is required	Description
port	Yes	Port number specified for HTTP Server connection on the **Web server** object settings panel (see Setting the parameters of connecting Clients to the Web-server).
hmac	Yes	Earlier received token

14.10.7.11 **Request example**:

GET http://127.0.0.1:80/action.do?hmac=GAqUa429sjY2E9jCTpuYeaMqReW3Y7HI

14.11 PTZ control

14.11.1 General request format:

GET http://{login}:{password}@IP-address:[port]/web2/secure/video/action.do?version={version}&sessionid={session_id}
&cam.id={cam_id}&target=PTZ&targetid={PTZ_device_id}&command={command}&login={login}&password={password}
&speed={speed}&preset={preset}

> **Important!**
>
> Login and password must be specified twice in the request: before the IP address and in parameters. In both cases, this is the same login and password of the *Intellect* user.

14.11.2 **Request parameters**:

Parameter	Is required	Description
version	Yes	See Product version
session_id	Yes	Session ID
cam_id	Yes	Camera ID
PTZ_device_id	Yes	ID of the PTZ control panel related to the camera (can be obtained in configuration request – see Getting list of all server objects)
command	Yes	The command to be executed. It can take one of the following values: • RIGHT – pan right • UP – tilt up • LEFT – pan left • DOWN – tilt down • ZOOM_IN – zoom in • ZOOM_OUT – zoom out • GO_PRESET – go to the specified preset • POINTMOVE – zooming of selected point on the image (x,y) • AREAZOOM – zooming of selected area on the image (x,y,w,h)
login	Yes	*Intellect* username, if set
password	Yes	*Intellect* user password, if set
speed	Yes	Command-processing speed (from 0 to 10). It is recommended to use low values due to delays while controlling by network
preset	No	Number of preset. Required parameter only for the command=GO_PRESET. Otherwise its value is ignored. **x** – x coordinate relatively the image size. It takes values from 0.0 to 1.0. Required parameter only for the command=POINTMOVE or command=AREAZOOM. Otherwise its value is ignored. **y** – y coordinate relatively the image size. It takes values from 0.0 to 1.0. Required parameter only for the command=POINTMOVE or command=AREAZOOM. Otherwise its value is ignored. **w** – width of zooming area relatively the image size. It takes values from 0.0 to 1.0. Required parameter only for the command=AREAZOOM. Otherwise its value is ignored. **h** – height of zooming area relatively the image size. It takes values from 0.0 to 1.0. Required parameter only for the command=AREAZOOM. Otherwise its value is ignored.

14.11.3 Request example:

GET http://user:pass@127.0.0.1:8085/web2/secure/video/action.do?version=4.10.0.0&cam.id=5cam.id=1&target=PTZ&targetid=1.1&command=RIGHT&login=user&password=pass&speed=2

Camera managing

14.12 Using the archive

Video archive stream is sent on the same format as live video.

14.12.1 Getting list of records (1st way)

14.12.1.1 General request format:

GET http://IP-address:port/web2/secure/video/action.do?version={version}&sessionid={sessionid}&video_in={video_in}
&command=arc.intervals&time_from={time_from}&time_to={time_to}&max_count={max_count}
&split_threshold={split_threshold}&login={login}&password={password}

14.12.1.2 Request parameters:

Parameter	Is required	Description
version	Yes	See Product version
sessionid	No	Session ID
video_in	Yes	Camera ID in the format "TYPE:ID", for example "CAM:1"
command	Yes	Command to receive list of records: arc.intervals
time_from	Yes	Start of interested time range
time_to	No	End of interested time range
max_count	No	Maximal number of records in reply
split_threshold	No	Time for combining several intervals (in seconds). Intervals, distance between which will be less than specified value, will be combined in one.
login	No	*Intellect* username, if set
password	No	*Intellect* user password, if set
format	No	Sets the response format (see General information on HTTP API)

14.12.1.3 **Request example**:

GET http://127.0.0.1:8085/web2/secure/video/action.do?
version=4.9.0.0&sessionid=29101F1&video_in=CAM:5&command=arc.intervals&time_from=2013-03-20T00:00:00.000+04:00&time_to=2013-03-22T23:59:59.999+04:00&max_count=100&split_threshold=10399&login=USER&password=PASS

14.12.1.4 Response **example**:

XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<records count="1" complete="YES" sort="INCREASE">
    <record>
        <from>2011-09-01T00:00:00-05:00</from>
        <to>2011-09-01T00:00:35-05:00</to>
    </record>
    <record>
        <from>2011-09-01T00:00:35-05:00</from>
        <to>2011-09-01T00:01:10-05:00</to>
    </record>
</records>
```

JSON:

```
{ 'count' : 1,
  'complete' : 'YES',
  'sort' : 'INCREASE',
  'cam' : '1',
  'records' : [ {
     'from' : '2019-01-22T12:41:10.144+03:00',
    'to' : '2019-01-23T08:28:47.346+03:00'
  }]
}
```

14.12.2 Getting list of records (2nd way)

14.12.2.1 General request format:

GET http://IP-address:port/web2/secure/archive/{CAM:id}/{DATE}/?[splitThreshold={splitThreshold}]&[days={days}]

14.12.2.2 Request parameters:

Parameter	Is required	Description
CAM:id	Yes	Camera ID in the format "TYPE:ID", for example "CAM:1"
DATE	Yes	Date of the start of receiving the archive. All time is interpreted as local time for server
splitThreshold	Yes	If difference between end of previous record and start of next record less than specified value (in milliseconds), than records will be combined in one. Specify splitTreshold=0 not to combine records. [default: 50]
days	Yes	Number of days from the current, for which archive is required. [default: 1]

14.12.2.3 **Request example**:

GET http://127.0.0.1:8085/web2/secure/archive/CAM:2/2011-12-30/?[splitThreshold=50]&[days=1]

GET http://127.0.0.1:8085/web2/secure/archive/CAM:1/2013-11-18/?splitTreshold=2000 – Get records for 18 November 2013 and combine all records, interval between which less than 2000 milliseconds.

GET http://127.0.0.1:8085/web2/secure/archive/CAM:1/2013-11-18/?days=10 – Get records for 10 days from 18 November 2013.

14.12.2.4 Response **example**:

XML:

```
<?xml version="1.0" encoding="UTF-16"?>
<days>
    <day>
        <id>2013-11-10T00:00:00-02:00</id>
        <records>
            <from>2013-11-10T18:44:01.579-02:00</from>
            <to>2013-11-10T18:44:09.717-02:00</to>
        </records>
    </day>
    <day>
```

```
        <id>2013-11-18T00:00:00-02:00</id>
        <records>
            <from>2013-11-18T18:38:30.252-02:00</from>
            <to>2013-11-18T18:38:56.942-02:00</to>
        </records>
        <records>
            <from>2013-11-18T18:39:08.321-02:00</from>
            <to>2013-11-18T18:39:10.080-02:00</to>
        </records>
    </day>
</days>
```

JSON:

```
[ {
  "id" : "2013-11-10T00:00:00.000-02:00",
  "records" : [ {
    "from" : "2013-11-10T18:44:01.579-02:00",
    "to" : "2013-11-10T18:44:09.717-02:00"
  } ]
}, {
  "id" : "2013-11-18T00:00:00.000-02:00",
  "records" : [ {
    "from" : "2013-11-18T18:38:30.252-02:00",
    "to" : "2013-11-18T18:38:56.942-02:00"
  }, {
    "from" : "2013-11-18T18:39:08.321-02:00",
    "to" : "2013-11-18T18:39:10.080-02:00"
  } ]
} ]
```

Getting records for a month (shows days of September at which there are records):

GET http://127.0.0.1:8085/web2/secure/archive/CAM:2/2011-09/

XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<days>
    <day>
        <id>2011-09-02T00:00:00-05:00</id>
    </day>
    <day>
        <id>2011-09-03T00:00:00-05:00</id>
    </day>
    <day>
        <id>2011-09-05T00:00:00-05:00</id>
    </day>
</days>
```

JSON:

```
[ {
  "id" : "2011-09-01T00:00:00-0500",
  "records" : [ ]
}, {
  "id" : "2011-09-03T00:00:00-0500",
```

```
  "records" : [ ]
}, {
  "id" : "2011-09-01T00:00:00-0500",
  "records" : [ ]
} ]
```

If there are no records, the following will be received:

XML:

```
[<days/>
JSON:
[]
```

14.12.3 Getting video from archive - "arc.play"

14.12.3.1 General request format:

GET http://IP-address:port/web2/secure/video/action.do?version={version}&sessionid={sessionid}&video_in={video_in}
&command=arc.play&time_from={time_from}&time_to={time_to}&login={login}&password={password}
&speed_factor={speed_factor}

14.12.3.2 Request parameters:

Parameter	Required	Description
version	Yes	See Product version
sessionid	No	Session ID
video_in	Yes	Camera ID in the format "TYPE:ID", for example "CAM:1"
command	Yes	Command to get video from archive: arc.play
time_from	Yes	Start time of archive playing back
time_to	No	Completion time of archive playing back (if parameter is not specified, all archive will playback)
imageWidth	No	Width in pixels (it is counted automatically with saving proportions if it isn't specified or equal 0)
imageHeight	No	Height in pixels (it is counted automatically with saving proportions if it isn't specified or equal 0)
fps	No	Maximal frame per second (if it isn't specified or equal 0, frame frequency won't be limited)
login	No	*Intellect* username, if set
password	No	*Intellect* user password, if set
speed_factor	No	Sets the playback speed. The value may be any integer or fractional number no less than 0. Examples of values: 0 – playback at the highest possible speed (depends on network bandwidth and disk load) 1 – playback at normal speed x1 (default) 0.1 – slow motion playback at x1/10 speed 2 – fast playback at x2 speed

Parameter	Required	Description
format	No	Sets the response format (see General information on HTTP API)

14.12.3.3 Request example:

GET http://127.0.0.1:8085/web2/secure/video/action.do?
version=4.9.0.0&sessionid=29101F1&video_in=CAM:5&
command=arc.play&time_from=2013-03-22T13:04:52.312+04:00&time_to=2013-03-22T13:16:31.873+04:00&
login=USER&password=PASS&speed_factor=1

14.12.3.4 Response example:

End packet with newstate=closed and errcode=100 will be received when stream completion.

> **Note**
>
> The request can be executed through the web or in the VLC media player.

14.12.4 Getting one frame from archive - "arc.frame"

14.12.4.1 General request format (1st way):

GET http://IP-address:port/web2/secure/video/action.do?version={version}&sessionid={sessionid}&video_in={video_in}
&command=arc.frame&time={time}&range={range}&login={login}&password={password}

14.12.4.2 Request parameters:

Parameter	Is required	Description
version	Yes	See Product version
sessionid	No	Session ID
video_in	Yes	Camera ID in the format "TYPE:ID", for example "CAM:1"
command	Yes	Command to get video from archive: arc.frame
time	Yes	Frame time
range	No	Time in seconds to specify search range of the nearest frame relatively the time parameter (the nearest frame all over archive is searched if this parameter is not specified);
imageWidth	No	Width in pixels (is counted automatically with saving proportions if it isn't specified or equal 0)
imageHeight	No	Height in pixels (is counted automatically with saving proportions if it isn't specified or equal 0)
login	No	*Intellect* username, if set
password	No	*Intellect* user password, if set

14.12.4.3 **Request example**:

GET http://127.0.0.1:8085/web2/secure/video/action.do?version=4.9.0.0&sessionid=29101F1&video_in=CAM:5&command=arc.frame&time=2013-03-22T13:04:52.312+04:00&range=0.1&login=USER&password=PASS

14.12.4.4 General request format (2nd way):

GET http://IP-address:port/action.do?version={version}&video_in={video_in}&command=arc.frame&time={time}

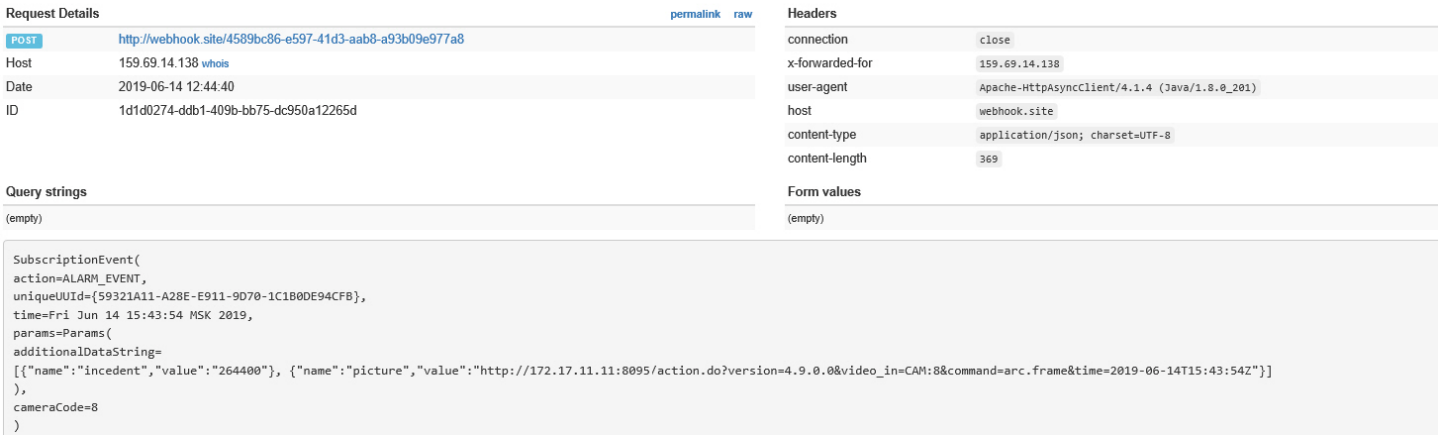
14.12.4.5 Request parameters:

Parameter	Is required	Parameter
version	Yes	See Product version
video_in	Yes	Camera ID in the format "TYPE:ID", for example "CAM:1"
command	Yes	Command to get video from archive: arc.frame
time	Yes	Frame time

14.12.4.6 **Request example**:

GET http://127.0.0.1:80/action.do?version=4.9.0.0&video_in=CAM:1&command=arc.frame&time=2018-08-12T22:29:06Z

14.12.4.7 Response **example**:



In return http-headings and the nearest frame from the [time - range, time + range] range in the jpeg format will be received. The reply body will be empty if there is no frame in the range.

14.13 Sound

14.13.1 Getting live sound

14.13.1.1 General request format:

GET http://IP-address:port/web2/secure/video/action.do?version={version}&sessionid={sessionid}&command=audio.play&audio_in={audio_in}&format={format}&login={login}&password={password}

14.13.1.2 Request parameters:

Parameter	Is required	Description
version	Yes	See Product version
sessionid	No	Session ID
command	Yes	Command to getting live sound: **audio.play**
format	Yes	Format of audio data
audio_in	Yes	Microphone ID in the format "TYPE:ID", for example "MIC:1"
login	No	*Intellect* username, if set
password	No	*Intellect user password, if set*

14.13.1.3 **Request example**:

GET http://127.0.0.1:8085/web2/secure/video/action.do?
version=4.9.0.0&sessionid=FC126734&command=audio.play&audio_in=MIC:5&format=L16&login=USER&password=PASS

14.13.1.4 Response **example**:

Audio packets of the following view will be received:

```
HTTP/1.0 200 OK
Connection: close
Server: Intellect-Webserver/4.9.0.0
Cache-Control: no-store,no-cache,must-revalidate,max-age=0
Pragma: no-cache
Date: Mon, 13 Jan 2013 10:44:27 GMT
Content-Type: multipart/mixed;boundary=audioframe

--audioframe
Content-Type: text/xml
Content-Length: 138

<audio_in>
  <sessionid>FC126734</sessionid>
  <audio_in>MIC:5</audio_in>
  <newstate>started</newstate>
  <errcode>100</errcode>
</audio_in>
--audioframe
Content-Type: audio/L16;rate=8000;channels=1
Content-Length: 1024
X-Time: 2013-03-22T13:16:31.371+04:00

<audio packet PCM16>
--audioframe
Content-Type: audio/L16;rate=8000;channels=1
Content-Length: 1278
X-Time: 2013-03-22T13:16:31.873+04:00
```

```
<audio packet PCM16>
```

14.13.2 Stop streaming live sound

14.13.2.1 General request format:

GET http://IP-address:port/web2/secure/video/action.do?version={version}&sessionid={sessionid}
&command=audio.stop&audio_in={audio_in}&login={login}&password={password}

14.13.2.2 Request parameters:

Parameter	Is required	Description
version	Yes	See Product version
sessionid	No	Session ID
command	Yes	Command to stop streaming live sound: **audio.stop**
audio_in	Yes	Microphone ID in the format "TYPE:ID", for example "MIC:1"
login	No	*Intellect* username, if set
password	No	*Intellect user password, if set*

14.13.2.3 **Request example**:

GET http://127.0.0.1:8085/web2/secure/video/action.do?
version=4.9.0.0&sessionid=29101F1&command=audio.stop&audio_in=MIC:5&login=USER&password=PASS

14.13.2.4 Response **example**:

The end xml packet will be received:

```
--audioframe
Content-Type: text/xml
Content-Length: 106
<audio_in>
  <sessionid>FC126734</sessionid>
  <audio_in>MIC:5</audio_in>
  <newstate>closed</newstate>
  <errcode>100</errcode>
 </audio_in>
```

14.13.3 Playing sound from archive

14.13.3.1 General request format:

GET http://IP-address:port/web2/secure/video/action.do?version={version}&sessionid={sessionid}
&command=arc.play&audio_in={audio_in}&format={format}&time_from={time_from}&time_to={time_to}&login={login}
&password={password}

14.13.3.2 Request parameters:

Parameter	Is required	Description
version	Yes	See Product version
sessionid	No	Session ID
command	Yes	Command to playing sound from archive: **arc.play**
format	Yes	Format of audio data
audio_in	Yes	Microphone ID in the format "TYPE:ID", for example "MIC:1"
time_from	Yes	Start time of archive playing back
time_to	Yes	End time of archive playing back
login	No	*Intellect* username, if set
password	No	*Intellect user password, if set*

14.13.3.3 **Request example**:

GET http://127.0.0.1:8085/web2/secure/video/action.do?
version=4.9.0.0&sessionid=29101F1&command=arc.play&audio_in=MIC:5&format=L16&time_from=2013-03-22T13:16:31.873+04:00&time_to=2013-03-22T13:04:52.312+04:00&login=USER&password=PASS

14.13.3.4 Response **example**:

The stream will be received in the same view as in case of live sound (see Getting live sound). The end xml packet will be received when data completion (as when getting live sound (see Stop streaming live sound).

14.13.4 Sending live sound

14.13.4.1 General request format:

POST http://IP-address:port/web2/secure/video/action.do?version={version}&sessionid={sessionid}
&command=audio.receive&audio_out={audio_out}&login={login}&password={password}

14.13.4.2 Request parameters:

Parameter	Is required	Description
version	Yes	See Product version
sessionid	No	Session ID
command	Yes	Command to sending live sound: **audio.receive**
audio_out	Yes	Speaker ID in the format "TYPE:ID", for example "SPEAKER:1"
login	No	*Intellect* username, if set

Parameter	Is required	Description
password	No	*Intellect user password, if set*

14.13.4.3 **Request example**:

POST http://127.0.0.1:8085/web2/secure/video/action.do?version=4.9.0.0&sessionid=FC126734&command=audio.receive&audio_out=SPEAKER:3&login=USER&password=PASS

14.13.4.4 Response **example**:

Sending of sound is performed by serial communication of packets using commands:

```
Content-type: audio/L16;rate=8000;channels=1
Connection: keep-alive
```

Then audio packet transmission will perform.

14.13.4.5 Response parameters:

Parameter	Description
L16	Format of sound – only L16
rate	Any rational value
channels	Channel from 1 to 6

14.14 Get list of users

14.14.1 General request format:

GET http://IP-address:port/web2/secure/persons

14.14.2 Request example:

GET http://127.0.0.1:8085/web2/secure/persons

14.14.3 Response example:

```
{
    "PERSON":
    {
        "external_id":
        {
            "type": "VarWChar",
            "value": ""
        },
        "auto_pass_type":
        {
            "type": "Integer",
            "value": ""
        },
        "rubeg8_zone_id":
```

```
        {
            "type": "VarWChar",
            "value": ""
        },
        "levels_times":
        {
            "type": "LongVarWChar",
            "value": ""
        },
        "expired":
        {
            "type": "VarWChar",
            "value": ""
        },
        "card_date":
        {
            "type": "VarWChar",
            "value": "19.03.2021 15:34:07"
        },
        "who_level":
        {
            "type": "VarWChar",
            "value": ""
        },
        "visit_purpose":
        {
            "type": "VarWChar",
            "value": ""
        },
        "level2_id":
        {
            "type": "VarWChar",
            "value": ""
        },
        "card":
        {
            "type": "VarWChar",
            "value": ""
        },
        "objname":
        {
            "type": "VarWChar",
            "value":
"\u00CF\u00EE\u00EB\u00FC\u00E7\u00EE\u00E2\u00E0\u00F2\u00E5\u00EB\u00FC 1"
        },
        "email":
        {
            "type": "VarWChar",
            "value": ""
        },
        "area_id":
        {
            "type": "VarWChar",
            "value": ""
        },
        "who_card":
        {
```

```
    "type": "VarWChar",
    "value": ""
},
"surname":
{
    "type": "VarWChar",
    "value": ""
},
"facility_code":
{
    "type": "VarWChar",
    "value": ""
},
"card_loss":
{
    "type": "Integer",
    "value": ""
},
"auto_brand":
{
    "type": "VarWChar",
    "value": ""
},
"when_area_id_changed":
{
    "type": "DBTimeStamp",
    "value": ""
},
"post":
{
    "type": "VarWChar",
    "value": ""
},
"drivers_licence":
{
    "type": "VarWChar",
    "value": ""
},
"temp_levels_times":
{
    "type": "LongVarWChar",
    "value": ""
},
"temp_level_id":
{
    "type": "LongVarWChar",
    "value": ""
},
"temp_card":
{
    "type": "VarWChar",
    "value": ""
},
"patronymic":
{
    "type": "VarWChar",
    "value": ""
```

```
},
"location":
{
    "type": "VarWChar",
    "value": ""
},
"teleph_work":
{
    "type": "VarWChar",
    "value": ""
},
"department":
{
    "type": "VarWChar",
    "value": ""
},
"parent_id":
{
    "type": "VarWChar",
    "value": "1"
},
"tabnum":
{
    "type": "VarWChar",
    "value": ""
},
"pur":
{
    "type": "Integer",
    "value": ""
},
"finished_at":
{
    "type": "DBTimeStamp",
    "value": ""
},
"all_param":
{
    "type": "VarWChar",
    "value": ""
},
"whence":
{
    "type": "VarWChar",
    "value": ""
},
"schedule_id":
{
    "type": "VarWChar",
    "value": ""
},
"person":
{
    "type": "VarWChar",
    "value": ""
},
"passport":
```

```
{
    "type": "VarWChar",
    "value": ""
},
"ngp":
{
    "type": "Integer",
    "value": ""
},
"flags":
{
    "type": "Integer",
    "value": ""
},
"auto_number":
{
    "type": "VarWChar",
    "value": ""
},
"pin":
{
    "type": "VarWChar",
    "value": ""
},
"phone":
{
    "type": "VarWChar",
    "value": ""
},
"begin_temp_level":
{
    "type": "VarWChar",
    "value": ""
},
"visit_document":
{
    "type": "VarWChar",
    "value": ""
},
"visit_birthplace":
{
    "type": "VarWChar",
    "value": ""
},
"is_apb":
{
    "type": "SmallInt",
    "value": "0"
},
"end_temp_level":
{
    "type": "VarWChar",
    "value": ""
},
"name":
{
    "type": "VarWChar",
```

```
            "value":
"\u00CF\u00EE\u00EB\u00FC\u00E7\u00EE\u00E2\u00E0\u00F2\u00E5\u00EB\u00FC 1"
        },
        "started_at":
        {
            "type": "DBTimeStamp",
            "value": ""
        },
        "level_id":
        {
            "type": "LongVarWChar",
            "value": ""
        },
        "_marker":
        {
            "type": "LongVarWChar",
            "value": ""
        },
        "where_area_id_ap_type":
        {
            "type": "VarWChar",
            "value": ""
        },
        "owner_person_id":
        {
            "type": "VarWChar",
            "value": ""
        },
        "is_locked":
        {
            "type": "SmallInt",
            "value": "0"
        },
        "is_active_temp_level":
        {
            "type": "SmallInt",
            "value": "0"
        },
        "guid":
        {
            "type": "GUID",
            "value": "{676A1F65-AF88-EB11-9633-1831BF4C3BFA}"
        },
        "card_type":
        {
            "type": "Integer",
            "value": ""
        },
        "begin":
        {
            "type": "VarWChar",
            "value": ""
        },
        "where_area_id_ap_id":
        {
            "type": "VarWChar",
            "value": ""
```

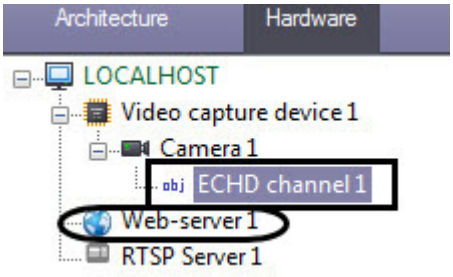
```
        },
        "visit_reg":
        {
            "type": "VarWChar",
            "value": ""
        },
        "marketing_info":
        {
            "type": "LongVarWChar",
            "value": ""
        },
        "comment":
        {
            "type": "LongVarWChar",
            "value": ""
        },
        "ad_guid":
        {
            "type": "GUID",
            "value": ""
        },
        "id": "1"
    }
```


14.15 Commands used for ECHD integration

ECHD means ‘Unified data center’ state information system.

The description of http requests that are used for integration of *Intellect* with ECHD is given in this section. For requests operation they are to be enabled at the stage of system configuration – see Enabling the processing of ECHD requests and selecting rtsp server.

Requests are sent to **Web Server** and are performed only for cameras under which the **ECHD channel** object is created.



14.15.1 Retrieving Device Information

14.15.1.1 General request format:

GET http://IP-address:port/getdeviceinfo

14.15.1.2 **Request example**:

GET http://127.0.0.1:80/getdeviceinfo

14.15.1.3 Response **example**:

```
{
    "firmware version": "1.2.3 Rev B.",
    "vendor": "Vendor Title Ltd"
    "model": "Device Model",
    "serial_number": "12345ABCDEF",
    "ptz-status": "not supported"
}
```

14.15.1.4 **Response** parameters:

Parameter	Description
firmware version	Firmware version. Any prefix can be added to the firmware version value returned by the AdditionalVersionString registry key – see Registry keys reference guide.
vendor	Manufacturer
model	Model
serial_number	Serial number
ptz-status	PTZ support

14.15.2 Getting a list of camera IDs

14.15.2.1 General request format:

GET http://IP-address:port/getcameras

14.15.2.2 **Request example**:

GET http://127.0.0.1:80/getcameras

14.15.2.3 Response **example**:

```
{
    "cameras": [
        {
            "id": 1,
            "channel": 1,
            "status": "working"
        },
        {
            "id": 2,
            "channel ": 2,
            "status": "signaIJost"
        }
            ]
}
```

14.15.2.4 **Response** parameters:

Parameter	Description
id	Camera ID
channel	Camera channel
status	Camera status

14.15.3 Ranges of available archive recordings

14.15.3.1 General request format:

GET http://IP-address:port/getarchiveranges?cameraid={cameraid}

or

GET http://IP-address:port/getavailablearchiveranges?cameraid={cameraid}

14.15.3.2 Request parameters:

Parameter	Is required	Description
cameraid	Yes	Camera ID

14.15.3.3 **Request example**:

GET http://127.0.0.1:80/getarchiveranges?cameraid=1

14.15.3.4 Response **example**:

Returns time periods over which archive recordings from the specified video surveillance device are available. By default fragments are merged if the interval between them is less than 5 seconds. This time period can be changed using the SplitArchiveIntervals registry key (see Registry keys reference guide).

```
{
    "cameraid": 1,
    "ranges": [
        {
            "from": 1412121600, //unixtime
            "to": 1412172000
        },
        {
            "from": 1412186400,
            "to": 1412188200
        }
            ]
}
```

14.15.4 Working with video streams

14.15.4.1 GetArchiveURL request

GetArchiveURL – returns rtsp URL of archive video stream received from video encoder for the specified camera starting from FromDateTime (and, optionally, ending at endDatetime).

14.15.4.1.1 General request format:

GET http://IP-address:port/getarchiveurl?cameraid={cameraid}&fromdatetime={fromdatetime}&todatetime={todatetime}

14.15.4.1.2 Request parameters:

Parameter	Is required	Description
cameraid	Yes	Camera ID
fromdatetime	Yes	Start time of the archive fragment in the format YYYY-MM-DDTHH:MM:SS
todatetime	No	End time of the archive fragment in the format YYYY-MM-DDTHH:MM:SS (if not specified, the entire archive will be given until the last record)

14.15.4.1.3 **Request example**:

GET http://127.0.0.1:80/getarchiveurl?cameraid=1&fromdatetime=2014-10-01T00:00:00&todatetime=2014-10-01T01:20:05

14.15.4.1.4 Response **example**:

```
{
"rtspurl": "rtsp://deviceladdress/somearchivemediastream?somedatetimetoken"
}
```

14.15.4.2 GetLiveUrl request

GetLiveUrl - returns rtsp URL of “live” video stream for the specified camera.

14.15.4.2.1 General request format:

GET http://IP-address:port/getliveurl?cameraid={cameraid}

14.15.4.2.2 Request parameters:

Parameter	Is required	Description
cameraid	Yes	Camera ID

14.15.4.2.3 **Request example**:

GET http://127.0.0.1:80/getliveurl?cameraid=1

14.15.4.2.4 Response **example**:

```
{
```

```
"rtspurl": "rtsp://device-address/somelivemediastream0"
}
```

14.15.5 Archive downloading

14.15.5.1 General request format:

GET http://IP-address:port/downloadarchivefile?cameraid={cam_id}&fromdatetime={from_time}&todatetime={to_time}

14.15.5.2 Request parameters:

Parameter	Is required	Description
cam_id	Yes	Camera ID
from_time	Yes	The start time of the archive fragment in the format YYYY-MM-DDTHH:MM:SS
to_time	Yes	The end time of the archive fragment in the format YYYY-MM-DDTHH:MM:SS

14.15.5.3 Request example:

GET http://127.0.0.1:80/downloadarchivefile?cameraid=1&fromdatetime=2014-10-01T00:00:00&todatetime=2014-10-01T01:20:05

14.15.5.4 Response example:

```
HTTP/l.1 200 OK
Content-Type: application/octet-stream
```

Also, as a result of the command, a file with the .es extension will be received (for example, Camera [4] (2019-08-13T11_00_00 - 2019-08-13T12_10_00).es). Convert this file using the ffmpeg utility to play it. This utility is available for download on the official website https://ffmpeg.org/

Example command to convert H.264-coded file:

ffmpeg -i "C:\path to the .264 file\Camera[5].es" -c:v copy -bsf:v h264_mp4toannexb -c:a copy -f avi output.avi

Example command to convert H.265-coded file:

ffmpeg -i "C:\path to the .265 file\Camera[5].es" -c:v copy -bsf:v hevc_mp4toannexb -c:a copy -f avi output.avi

The .avi file (output.avi) is created after this command execution.

14.15.6 Archive export

14.15.6.1 Creating archive export request

By default, the archive is exported in mp4 format. Change the format using the ExportContainerFormat registry key (see Registry keys reference guide). The export of the archive in H.264 or MPEG4 formats is supported.

14.15.6.1.1 General request format:

POST http://IP-address:port/createarchivetask

```
Content Type: application/json
Content:
{
 "CameraId": "{CameraId}",
 "From": "{From}",
 "To": "{To}"
}
```

14.15.6.1.2 Request parameters:

Parameter	Is required	Description
CameraId	Yes	Camera ID
From	Yes	The start time of the archive fragment in the UTC format YYYY-MM-DDTHH:MM:SS
To	No	The end time of the archive fragment in the UTC format YYYY-MM-DDTHH:MM:SS (if not specified, the entire archive will be given up to the last record)

14.15.6.1.3 **Request example**:

POST http://127.0.0.1:80/createarchivetask

```
Content Type: application/json
Content:
{
 "CameraId": "1",
 "From": "2016-06-27T15:10:00.00Z",
 "To": "2016-06-27T15:20:00.00Z"
}
```

14.15.6.1.4 Response **example**:

The folder with the corresponding name (084b56a5-bd49-4327-82db-9bc911f7ff96) and mp4 file inside it is created in the export folder (by default C:\Users\User\Documents\Intellect\export).

```
{
    "CameraId" : "1",
    "From" : "2016-06-27T15:10:00.00Z",
    "To" : "2016-06-27T15:20:00.00Z",
    "ArchiveTaskId" : "084b56a5-bd49-4327-82db-9bc911f7ff96",
    "ErrorMessage" : null,
    "State" : "Created"
}
```

14.15.6.1.5 Response **example**:

Parameter	Description
CameraId	Camera ID

Parameter	Description
From	The start time of the archive fragment in the UTC format YYYY-MM-DDTHH:MM:SS
To	The end time of the archive fragment in the UTC format YYYY-MM-DDTHH:MM:SS
ArchiveTaskId	Task ID
ErrorMessage	Error messages
State	Task creation result

14.15.6.2 Getting export status

14.15.6.2.1 General request format:

GET http://IP-address:port/getarchivetaskstatus?archivetaskid={archivetaskid}

14.15.6.2.2 Request parameters:

Parameter	Is required	Description
archivetaskid	Yes	Task ID

14.15.6.2.3 **Request example**:

GET http://127.0.0.1:80/getarchivetaskstatus?archivetaskid=104b38d4-07d7-4d2f-84da-49b3e255d2bf

14.15.6.2.4 Response **example**:

```
{
    "Percents" : 100,
    "Url" : "http://192.168.15.182:80/download?file=104b38d4-07d7-4d2f-84da-49b3e255d2bf",
    "CameraId" : "1",
    "From" : "2016-06-27T15:10:00.000+03:00",
    "To" : "2016-06-27T15:11:00.000+03:00",
    "ArchiveTaskId" : "104b38d4-07d7-4d2f-84da-49b3e255d2bf",
    "ErrorMessage" : "null",
    "State" : "ReadyForDownload"
}
```

14.15.6.2.5 **Response** parameters:

Parameter	Description
Percents	Percentage of task completion
Url	Link to mp4 file. The required mp4 file can be downloaded using this URL link
CameraId	Camera ID
From	The start time of the archive fragment in the format YYYY-MM-DDTHH:MM:SS

Parameter	Description
To	The end time of the archive fragment in the format YYYY-MM-DDTHH:MM:SS
ArchiveTaskId	Task ID
ErrorMessage	Error messages
State	Task creation result

14.15.6.3 Deleting archive

14.15.6.3.1 General request format:

DELETE http://IP-address:port/removearchive?archivetaskid={archivetaskid}

14.15.6.3.2 Request parameters:

Parameter	Is required	Description
archivetaskid	Yes	Task ID

14.15.6.3.3 **Request example**:

DELETE http://127.0.0.1:80/removearchive?archivetaskid=084b56a5-bd49-4327-82db-9bc911f7ff96

14.15.6.3.4 Response **example**:

```
{
    "ArchiveTaskId" : "084b56a5-bd49-4327-82db-9bc911f7ff96",
    "ErrorMessage" : null,
    "Success" : true
}
```

14.15.6.3.5 **Response** parameters:

Parameter	Description
ArchiveTaskId	Task ID
ErrorMessage	Error messages
Success	Archive deletion result

The corresponding folder with mp4 file will be deleted from the export folder.

14.15.7 Video surveillance device features management

Important!

Disable the **Use device settings** checkbox in order to change the manage parameters by the below commands – see The Settings panel of the Video capture device object.
A camera must support the same commands for features management as the given ECHD commands (both when connected via ONVIF and via the corresponding driver).

14.15.7.1 General request format:

GET https://IP-address:port/?cameraID={1}&ip={2}&loqin={3}&pass={4}&action={5}&x={6}&y={7}&z={8}&modelName={9}

14.15.7.2 Request parameters:

Parameter	Is required	Description
1	Yes	ID of video surveillance device.
2	No	IP address of video surveillance device.
3	No	account of video surveillance device.
4	No	access password to video surveillance device.
5	Yes	Command name: **degreesmove** – discrete motion. Atomic shift of video surveillance device in the specified direction. **degreesmove2** – relative motion. Rotation of video surveillance device compared with current position. Viewing area of video surveillance device is divided by a grid where central point's coordinates are (x:0, y:0), top left (x:-7, y:7), bottom right (x:7, y:-7). Video surveillance device must be rotated in a way that object appears in the center of the grid. 'Optical' error caused by the distance to the visible object is allowed. Error caused by sphere-to-plane projection must be compensated. Depending on camera, registry keys may have to be set for correct operation: 1. The camera does not support Point&Click but supports absolute PTZ. Set the ReplacePointAndClick registry key to 1 (see Registry keys reference guide).

Parameter	Is required	Description
		2. The camera supports Point&Click. Set the ReplacePointAndClick registry key to 0 and TelemetryCommandMoveTimeout to delay in milliseconds between panning/tilting and zooming (see Registry keys reference guide). **setposition** – setting position of video surveillance device in degrees compared with '0' position. **getposition** – getting position of video surveillance device in pAN and TILt in degrees as well as current zoom values. **focus** – video surveillance device's focus control command where z parameter controls over focus: 1: Focus in -1 : Focus out 0: Auto **iris** – video surveillance device's iris control command where z parameter controls over iris: 1: Open iris -1: Close iris 0: Auto **switch_day_night** – switch day/night mode where z parameter enables one of the following modes: 1: Day mode -1 : Night mode **backlight** – switch backlight on/off where z parameter controls over the mode: 1: Enable -1: Disable **switch_color** – the following operation modes of video surveillance device are set by z parameter: 1: Enable -1: Disable
6	No	In **degreesmove, setposition** commands: PAN rotation [-180 ..0.. 180]. In **degreesmove2**: PAN rotation [-7..0..7]. In **getposition:** not in use. In commands **focus, iris, switch_day_night, backlight, switch_color:** set value 0 to the parameter.
7	No	In **degreesmove, setposition** commands: TILT rotation [-180 ..0.. 180]. In **degreesmove2**: TILT rotation [-7..0..7]. In **getposition:** not in use. In commands **focus, iris, switch_day_night, backlight, switch_color:** set value 0 to the parameter.
8	No	In **degreesmove, degreesmove2, setposition** commands: zoom in/out [0.. 100]. In **getposition:** not in use. In commands **focus, iris, switch_day_night, backlight, switch_color:** set device mode, see description of the corresponding command above.
9	No	Model of video surveillance device.

14.15.7.3 Request example:

GET http://127.0.0.1:80/execute?cameraID=7&action=getposition

14.15.7.4 Response example:

Response comes on getposition command only. Example in JSON format:

{"y":56, "x":105, "z":0}

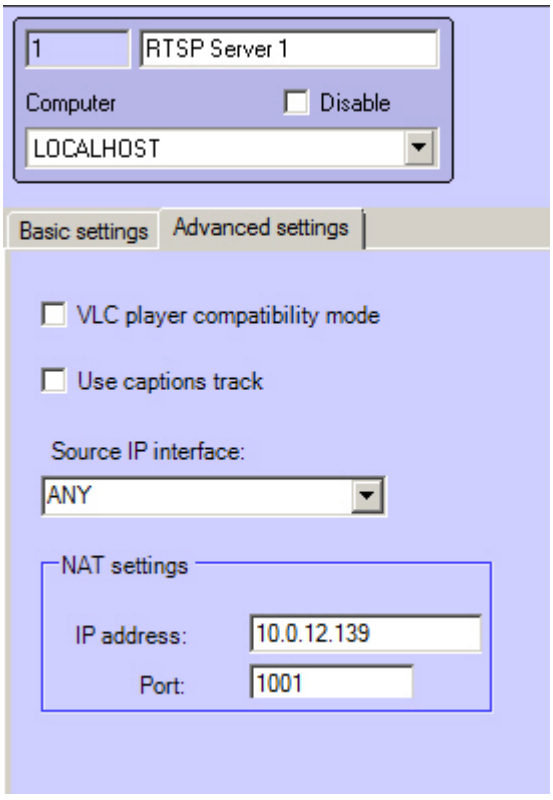
Response parameters:

Parameter	Description
x	PAN coordinate
y	TILT coordinate
z	Zoom value

14.15.8 Example ECHD commands with NAT

Go to the **RTSP Server** and **Web server** objects setting panels to enable and configure NAT (see Configuring RTSP Server module and Enabling the processing of ECHD requests and selecting the RTSP server).

For example, the following NAT parameters are specified on the **RTSP Server** object settings panel:



10.0.12.139 – router IP address.

1001 – router port.

192.168.0.109 – Server address in local network.

559 – RTSP Server port.

80 – Web Server port.

14.15.8.1 General request format:

GET http://IP-address:port/getliveurl?cameraid={cameraid}

14.15.8.2 Request parameters:

Parameter	Is required	Description
cameraid	Yes	Camera ID

14.15.8.3 **Request example**:

```
GET http://127.0.0.1:80/getliveurl?cameraid=1
```

14.15.8.4 Response **example**:

If the **Use NAT address** checkbox on the **Web server** object setting panel is UNchecked, then:

```
{
    "rtspurl": "rtsp://127.0.0.1:559/archive?id=1"
}
```

If the **Use NAT address** checkbox on the **Web server** object setting panel is checked, then:

```
{
    "rtspurl": "rtsp://10.0.12.139:1001/archive?id=1"
}
```

14.15.8.5 **Response** parameters:

Parameter	Description
id	Camera ID

14.16 Sending reactions, events or XML data to Intellect with HTTP API

14.16.1 Sending reactions to Intellect via HttpListener

In order to send reactions to *Intellect* via HttpListener, create the **Video Capture Device** object on the basis of the **Computer** object on the **Hardware** tab of the **System settings** dialog box. Then select **HttpListener** in the **Type** field and set connection port in the **Port** field. Create no more than 4 **Sensor** objects on the basis of this **Video Capture Device** object.

HttpListeners only allows sending close/open reactions to a normally opened/closed sensor. In 2 seconds after the command execution, the sensor goes back to its normal state. Macros or scripts can be configured in *Intellect* to handle sensor triggering.

14.16.1.1 General request format:

POST http://IP-address:port/device/id/{id}

```
{"state": "{state}"}
```

14.16.1.2 Request parameters:

Parameter	Required	Description
Port	Yes	HttpListener port (corresponds to the port number specified in the Web server settings)
id	Yes	Sensor ID: 0/1/2/3
state	Yes	Sensor state: • **opened** – Open • **closed** – Close

14.16.1.3 **Request example**:

POST http://127.0.0.1:8085/device/id/0

```
{"state": "closed"}
```

14.16.2 Sending events on scanned barcode

The HTTP API allows sending events to *Intellect* from a mobile application with the information received from a scanned barcode.

14.16.2.1 General request format:

POST http://IP-address:port/web2/secure/barcode/

```
{ "value": "value", "img":"img" }
```

14.16.2.2 Request parameters:

Parameter	Is required	Description
value	Yes	Barcode number
img	No	Identifier of the scanned image

14.16.2.3 Request example:

http://1:1@127.0.0.1:8085/web2/secure/barcode

```
{ "value": "321", "img":"6835" }
```

14.16.2.4 Response example:

```
Event : WEB2|1|WEB2_BARCODE|slave_id<Computer>,fraction<243>,owner<Tester>,date<03-12-21>,guid_
pk<{96CC2A50-3F54-EC11-B93B-F02F74F7DC0D}>,core_global<1>,barcode<321>,img<6835>,time<16:45:53>
```

14.16.3 Sending reactions and events to Intellect using HTTP request

When getting such commands, standard events and reactions will be generated in *Intellect.* They can be used in scripts and macros (see Creating and using macros and Programming Guide (JScript)).

14.16.3.1 General request format:

> **Note**
>
> Create and configure the **Web server** object in order to use these requests – see Creating the Web-server object. In the requests, it is necessary to specify the port set up in the Web server settings.

GET http://IP-address:port/intellect_core/React?command="{react}"
GET http://IP-address:port/intellect_core/Event?command="{event}"

or (with the same result)

POST http://IP-address:port/intellect_core/React HTTP/1.1

```
{
    "command" : "{react}"
}
```

POST http://IP-address:port/intellect_core/Event HTTP/1.1

```
{
    "command" : "{event}"
}
```

14.16.3.2 Request parameters:

Parameter	Required	Description
command	Yes	React – a reaction in the *Intellect* format Event – an event in the *Intellect* format

14.16.3.3 **Request example**:

Add captions to video from camera 2 via HTTP request:

```
GET http://127.0.0.1:80/intellect_core/React?command="CAM|2|ADD_SUBTITLES|command<Some text\n!
>"
```

Generate alarm on camera 2 via HTTP request:

```
GET http://127.0.0.1:80/intellect_core/Event?command="CAM|2|MD_START"
```

OR (with the same result)

```
POST http://127.0.0.1:80/intellect_core/Event HTTP/1.1
{
```

```
"command" : "CAM|2|MD_START"
}
```

Run Macro 1 via HTTP request:

```
GET http://127.0.0.1:80/intellect_core/React?command="MACRO|1|RUN"
```

Start recording on Camera 1:

```
POST http://127.0.0.1:80/intellect_core/React HTTP/1.1
{
"command" : "CAM|1|REC"
}
```

14.16.4 Sending HTTP API commands using curl tool

To test HTTP API one can send HTTP API commands using curl tool. This tool is available at https://curl.haxx.se/.

To use the tool start the Windows command line and go to <curl installation directory>\curl-7.46.0-win64\bin folder.

14.16.4.1 **Request example**:

Find the example of the command to create the archive export task below as in the case of a request to a Single Data Center (see Archive export):

curl -H "Content-Type: application/json" -X POST -d "{ \"CameraId\": \"1\", \"From\": \"2017-12-26T10:58:00.00Z\", \"To\": \"2017-12-26T11:00:00.00Z\" } " http://127.0.0.1:80/createarchivetask

14.16.4.2 Response **example**:

```
{
    "CameraId" : "1", "From" : "2017-12-26T10:58:00.00Z", "To" : "2017-12-26T11:00:00.00Z",
"ArchiveTaskId" : "084b56a5-bd49-4327-82db-9bc911f7ff96", "ErrorMessage" : null, "State" :
"Created"
}
```

> **Note**
>
> In the requests, it is necessary to specify the port set up in the Web server settings — see Configuring the Web-server module.

14.16.5 Sending XML file

The HTTP API allows receiving data in XML format for further processing in scripts. To send a file, perform a POST request.

14.16.5.1 General request format:

POST http://IP-address:port/intellect_core/{Any}

```
<tag1>
```

```
    <tag2>some_data</tag2>
    <tag3>another_data</tag3>
</tag1>
```

14.16.5.2 Request parameters:

Parameter	Required	Description
Any	Yes	Any set of valid characters can be specified, except for Event and React, for example: • http://127.0.0.1:8080/intellect_core • http://127.0.0.1:8080/intellect_core/Any • http://127.0.0.1:8080/intellect_core/Custom

> **Note**
>
> In the requests, it is necessary to specify the port set up in the Web server settings — see Configuring the Web-server module.

14.16.5.3 Request example:

POST http://127.0.0.1:8080/intellect_core/Any

```
<tag1>
    <tag2>some_data</tag2>
    <tag3>another_data</tag3>
</tag1>
```

14.16.5.4 Response example:

After receiving the data in XML format in *Intellect*, the following event is generated:

```
HTTP|1|CUSTOM_EVENT|url</intellect_core/Any>,owner<SLAVE-ID>,data<PG5..90ZT4=>
```

14.16.5.5 Response parameters:

Parameter	Description
data	Request body (XML in the example above) base64-encoded
url	Part of the posted url

14.17 Configuring the Technoserv integration

Create and configure the following objects and files in order to configure the Technoserv integration:

1. **Web-server** – see Configuring the videoserver to connect Clients via the Web-server module.
2. **Web-server 2.0** – see Configuring the Server to connect the Clients via the Web-server 2.0 module.
3. At least one **User** object added to **User permissions**. The Technoserv connection with *Intellect* will be established under this user's credentials. See Rights administration.
4. Enable filter for camera and detection tool events on the **Web-server 2.0** object settings panel – see Configuring the Events filter for the Web-server 2.0 module.

5. Add all cameras to get events from to the Web-server configuration – see Selecting and configuring cameras for the Web-server module.
6. Fill in the ApiBgConfig.xml configuration file, then put it to "<*Intellect* installation path>\Modules\Jetty\". See Filling in the ApiBgConfig.xml configuration file.

See also Examples of commands to work with the Technoserv integration.

14.17.1 Filling in the ApiBgConfig.xml configuration file

The ApiBgConfig.xml file is used to configure the login and password for connecting the Technoserv system to *Intellect*, as well as for associating *Intellect* events with the Technoserv event codes (see the table below).

The file has the following format:

```
<ApiBgEventsConfiguration>
    <PingTimeout>120000</PingTimeout>
    <MainHost>127.0.0.1</MainHost>
    <Login></Login>
    <Password></Password>
    <ApiBgEventMatch>
        <ProductObjectName>CAM</ProductObjectName>
        <ProductEventName>DETACHED</ProductEventName>
        <ApiBgEventCode>not_available</ApiBgEventCode>
    </ApiBgEventMatch>
    <ApiBgEventMatch>
        <ProductObjectName>CAM</ProductObjectName>
        <ProductEventName>ATTACH</ProductEventName>
        <ApiBgEventCode>available</ApiBgEventCode>
        <Picture>1</Picture>
    </ApiBgEventMatch>
    <ApiBgEventMatch>
        <ProductObjectName>CAM</ProductObjectName>
        <ProductEventName>ALARM</ProductEventName>
        <ApiBgEventCode>264400</ApiBgEventCode>
        <Shapes>1</Shapes>
        <Picture>1</Picture>
    </ApiBgEventMatch>
        <ApiBgEventMatch>
        <ProductObjectName>CAM</ProductObjectName>
        <ProductEventName>REC</ProductEventName>
        <ApiBgEventCode>264409</ApiBgEventCode>
        <Shapes>1</Shapes>
        <Picture>1</Picture>
    </ApiBgEventMatch>
</ApiBgEventsConfiguration>
```

1. The time in milliseconds between sending PING events
   <PingTimeout>120000</PingTimeout>
2. The IP-address to be returned on requests of Technoserv.
   <MainHost>192.168.0.171</MainHost>
3. Username and password to connect to the Technoserv server and send events (if any)
   <Login></Login> <Password></Password>
4. Correspondence of the *Intellect* event to the event code:

```
<ApiBgEventMatch>
     <ProductObjectName>CAM_VMDA_DETECTOR</ProductObjectName>
     <ProductEventName>ALARM</ProductEventName>
     <ApiBgEventCode>264400</ApiBgEventCode>
     <ProductObjectIds>
       <id>2</id>
       <id>1</id>
     </ProductObjectIds>
     <ParamMatch>
         <ProductEventParamName>num</ProductEventParamName>
        <ApiBgEventParamCode>NUMBER_OF_DETECTED_PEOPLE</ApiBgEventParamCode>
     </ParamMatch>
     <Shapes>1</Shapes>
     <Picture>1</Picture>
</ApiBgEventMatch>
```

   a. ProductObjectName, ProductEventName – Intellect object and event, for example, CAM|MD_START.
   b. ProductObjectIds – Intellect object id, if you need to specify specific detectors. May be absent, then events will be transmitted to all detectors.
   c. ApiBgEventCode – event code, see below.
   d. Shapes – whether to transfer the Shapes field.

e. Picture – whether to send a link to the freeze frame.
f. ParamMatch – correspondence of the Intellect event parameter name to the event parameter. May be absent.

Event codes:

Code	**Event name**
1799	Crowd Detection (detect crowds, including in unauthorized places)
264393	Abandoned object detection
264383	People counter
264385	Estimated people traffic density in significant city sites
264388	Detection of moving against crowd flow
264387	Person stopped in the controlled area (with the preset stop time)
264386	Sharp acceleration of human movement
264389	Chaotic human movement (idleness)
264391	Indexing events in traffic conditions (traffic jams)
264392	Indexing of events in traffic conditions (massive accumulation of vehicles)
264390	Indexing events in traffic conditions (traffic density)
264394	Detection of crossing the forbidden zone (by human or vehicle)
264395	Disappeared object detection
264396	Responding to the passage of people in a given direction (entrance)
264397	Responding to the passage of people in a given direction (exit)
264399	Responding to the passage of people in a given direction (corridor etc.)
264398	Responding to the passage of people in a given direction (crosswalk)
264400	Appearance of a person or car in the observation zone (streets, squares, intersections, parks)
1795	Focus loss detection
1797	Dirty camera lens
264403	Background change
264402	Camera shift

Code	Event name
1796	Blind/cover detection
264401	Bare flame
287069	Target detected
siren	Siren or car alarm detected
shout	Human cry detected
klaxon	The sound of the car horn detected
voice	Human speech detected
brake	The sound of a vehicle brake detected
shockwave	A shock wave detected
shock	A bang/shot/blast detected
PING	It is used for a regular event periodicity sent to control the communication channel between the IS and the "Safe City" software and hardware complex. Value is filled as cameraCode=-1.
not_available	The camera is unavailable (there is no video stream or no connection with the camera, etc.)
available	The camera is available

14.17.2 Examples of commands to work with the Technoserv integration

14.17.2.1 Getting a list of cameras

14.17.2.1.1 General request format:

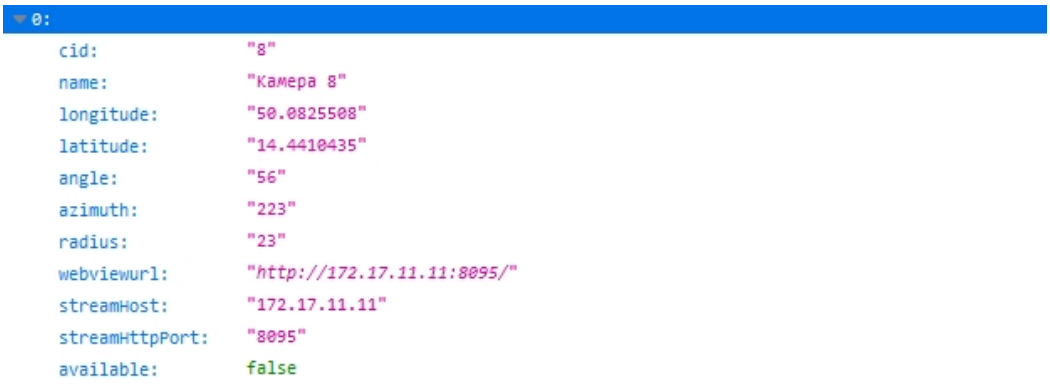
GET http://IP-address:port/web2/secure/api/bg

14.17.2.1.2 **Request example**:

GET http://127.0.0.1:8085/web2/secure/api/bg

> **Note.**
>
> If in the **Additional information** field on the camera settings panel is filled in the format coords:[longitude]:[latitude]:[angle]:[azimuth]:[radius], then this data will be returned in response to the request in the corresponding parameters. See also Additional information on camera.

14.17.2.1.3 Response **example**:



14.17.2.2 Get a list of subscriptions

14.17.2.2.1 General request format:

GET http://IP-address:port/web2/secure/api/bg/events/subscriptions

14.17.2.2.2 **Request example**:

GET http://127.0.0.1:8085/web2/secure/api/bg/events/subscriptions

14.17.2.2.3 Response **example**:

when the list is empty:

{"data": [], "status":"success"}

when the list has subscriptions:

14.17.2.3 Create subscription

14.17.2.3.1 General request format:

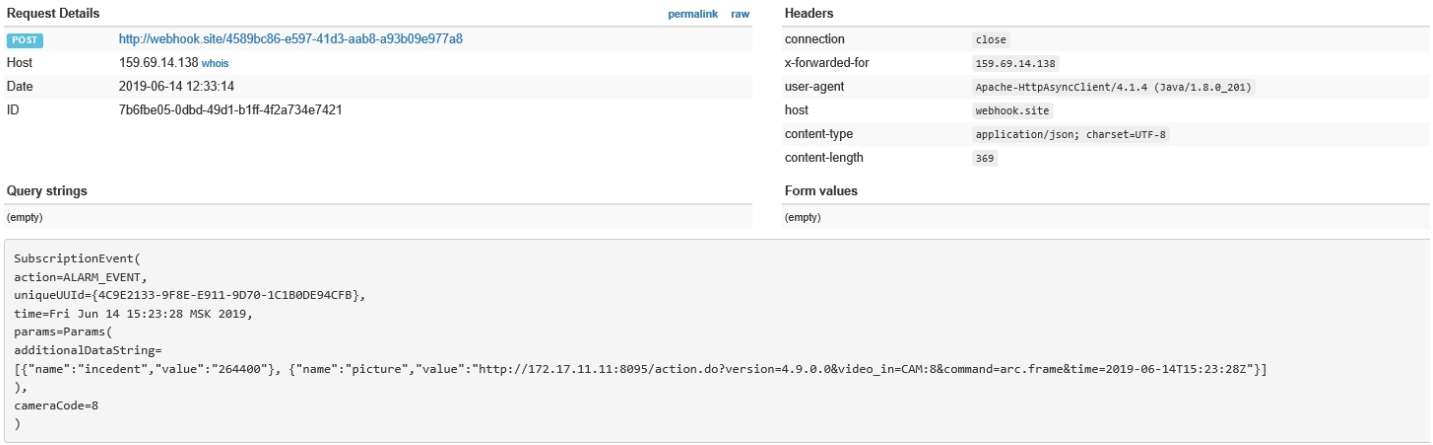
http://IP-address:port/web2/secure/api/bg/events/subscriptions

14.17.2.3.2 **Request example**:

POST http://127.0.0.1:8085/web2/secure/api/bg/events/subscriptions

{"callback":"https://webhook.site/26d15078-c405-4918-8f03-4f2a01b7580f","filter":{"action":"RUN","type":"MACRO"},"id":"5"}

14.17.2.3.3 Response **example**:



14.17.2.4 Delete subscription

14.17.2.4.1 General request format:

DELETE http://IP-address:port/web2/secure/api/bg/events/subscriptions/{sub_id}

14.17.2.4.2 **Request example**:

Parameter	Is required	Description
sub_id	Yes	Subscription ID

14.17.2.4.3 Response **example**:

DELETE http://127.0.0.1:8085/web2/secure/api/bg/events/subscriptions/071e0159-5b2c-4bab-8ed7-42397f1b99b8

14.17.2.5 Delete all subscriptions

14.17.2.5.1 General request format:

DELETE http://IP-address:port/web2/secure/api/bg/events/subscriptions/all

14.17.2.5.2 **Request example**:

DELETE http://127.0.0.1:8085/web2/secure/api/bg/events/subscriptions/all

14.18 Calling API of vertical solutions via Intellect HTTP API

The Intellect HTTP API allows you to access the Face-Intellect API or Auto-Intellect API via Web Server 2.0. To use this feature, you need the following:

- *Face-Intellect* REST API:
  - The **Face Recognition Server** object must be added to the Web server 2.0 filter (see Configuring the Events filter for the Web-server 2.0 module)

    - The response to the request for the list of objects must contain an object FIRSERVER type (see Getting list of all server objects)
- *Auto-Intellect* REST API:
    - The **LPR channel** object must be added to the Web server 2.0 filter (see Configuring the Events filter for the Web-server 2.0 module)
    - The response to a request for a list of objects must contain an object of the ULPR type (see Getting list of all server objects)

14.18.1 General request format

POST http://IP address:port/web2/secure/{vertical_solution}/{function}

14.18.2 Request parameters

Parameter	Is required	Description
vertical_solution	Yes	Specifies the vertical solution to access the API. Possible values: auto – *Auto Intellect* face – *Face Intellect* If **face** value is specified, then it is necessary to send the id of the Face Recognition Server to *Face Intellect* both in the URL and in the request body: POST http://IP-address:port/web2/secure/face/{server_id}/{function}
function	Yes	Vertical solution API function. See the description of the corresponding API.

Note.

Please refer to the documentation for the corresponding API function to see the parameters of the JSON sent in the POST request.

14.18.3 Request examples

POST http://1:1@127.0.0.1:8085/web2/secure/face/1/firserver/ReadPersons

{ "server_id": "1", "objectType": "PERSON", "id": [ ], "page": 1, "pageSize": 2 }

GET http://1:1@127.0.0.1:8085/web2/secure/lprserver/GetImage/Frames/66FB34A2-1B38-E811-A92F-001A7DDA710E

14.18.4 Response examples

Please refer to the documentation for the corresponding vertical solution API for the response examples.

15 Intellect HTTP-Server

15.1 General information on HTTP-Server

HTTP-Server allows you to transfer the events from *Intellect* to the third-party systems, i.e. to notify the clients about the events.

The *HTTP-Server* module operation requires the web browser to support the Server-sent events (SSE) technology.

> **Note**
>
> Internet Explorer browser does not support the SSE technology.

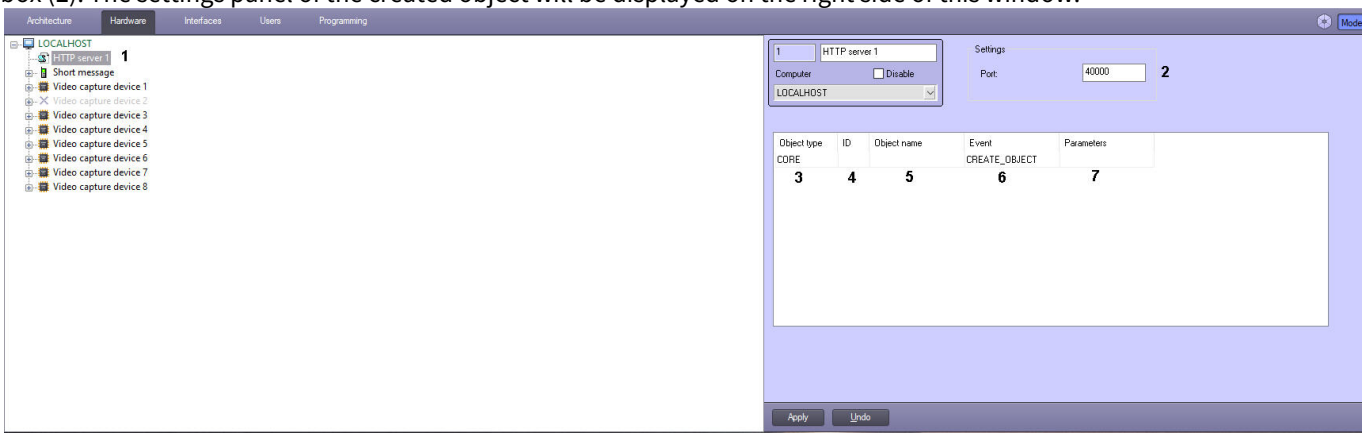
The *HTTP-Server* module is configured on the settings panel of the corresponding object — see Configuring the HTTP-Server object.

The requests sent from a web browser are used to work with the *HTTP-Server* module - see HTTP-Server requests.

15.2 Configuring the HTTP-Server object

The **HTTP-Server** object is configured as follows:

1. Create the **HTTP-Server** object on the basis of the **Computer** object on the **Hardware** tab of the **System Settings** dialog box (**1**). The settings panel of the created object will be displayed on the right side of this window.



2. In the **Port** field, specify the port number in the system where the client should send the *HTTP-Server* requests (**2**).
3. Configure the filter for the events which information should be sent by the *HTTP-Server*. If no events are added to the filter, then the *HTTP-Server* will return only the core events (the CORE object) for the client requests. The additional events are selected as follows:
    a. From the **Object type** drop-down list, select the required *Intellect* object type (**3**).
    b. From the **ID** drop-down list, select the identifier of the object of the selected type (**4**). If no ID is selected, then the *HTTP-Server* will send the events for all objects of the selected type.
    c. After selecting the type and object identifier, the **Object name** field will be automatically filled with the name of the corresponding object (**5**).
    d. In the **Event** field, enter the name of event which should be sent via the *HTTP-Server* (**6**). If no event is selected, the *HTTP-Server* will send all events of the selected object (or all objects of the selected type). The name of the event is specified in the internal format, for more details see Description of system object reactions. For example, set object type **Camera** and enter "ARM" in the **Event** field to filter camera arming events.
    e. If it is necessary to send selected events related only to users from the **Access Manager** module (see Access Manager Module Settings and Operation Guide), then enter **objtype<PERSON>** in the **Parameters** field (**7**). If this parameter is not specified, the events of all objects will be sent.
    f. Repeat steps a-e for all required objects and events.
4. Click **Apply**.

Configuring the **HTTP-Server** object is complete.

15.3 HTTP-Server requests

The requests described below are used when working with the *HTTP-Server* module.

The last event ID request

```
GET http://<IP address>:<Port>/core/GetLastID
```

Request example:

```
GET http://localhost:40000/core/GetLastID
```

Response example:

```
{"lastId":"b686658c-764c-e911-8f42-001a7dda710e"}
```

The event request based on filter

The filter is configured on the settings panel of the HTTP-Server object—see Configuring the HTTP-Server object.

```
GET http://<IP address>:<Port>/core/Events?keepAliveTime={keepAliveTime}
&startingID={startingID}
```

Request parameters:

Parameter	Required	Description
keepAliveTime	No	The time interval in seconds, within which the server will send a json of the {"time":"2019-03-19T14:56:26.317"} form to the client, containing the server time in UTC. The default is 5 seconds
startingID	No	ID of the event starting from which it is necessary to receive events

Request example:

```
GET http://localhost:40000/core/Events?keepAliveTime=5&startingID=8FFCDF07-5E4A-
E911-95AE-F894C2A95BA4
```

Response example:

```
{"module":"video.run","protocol_id":"2dc6dfcb-5351-e911-8832-534e57000000","slave_id":"S-
UYUTOVA","src_action":"MD_STOP","src_objid":"2","src_objtype":"CAM","time":
"2019-03-28T12:20:03.977"}
```

The request of the list of all created readers

GET http://<IP address>:<Port>/core/GetReaders

Request example:

http://localhost:22441/core/GetReaders

with the body:

```
{
"detail": true
}
```

Response example:

```
{
    "objects": [
        {
            "id": "1.1.1",
```

```
            "guid": "04EF2B52-90D2-EC11-9515-D8BBC1166DF4",
            "name": "ACS emulator 1.1.1 Reader",
            "_address": "1",
            "_marker": "",
            "flags": "",
            "objname": "ACS emulator 1.1.1 Reader",
            "objtype": "ACFA_EMULATOR_ACS_READER",
            "parent_id": "1.1",
            "parent_type": "ACFA_EMULATOR_ACS_CONTROLLER",
            "region_in": "",
            "region_out": ""
        },
        {
            "id": "1",
            "guid": "02FA79B0-A5D2-EC11-9515-D8BBC1166DF4",
            "name": "Hikvision Control Reader K1F100 1",
            "_marker": "",
            "card_save_mode": "0",
            "flags": "",
            "objname": "Hikvision Control Reader K1F100 1",
            "objtype": "HIK_CR_K1F100_D8E",
            "parent_id": "localhost",
            "parent_type": "SLAVE"
        },
        {
            "id": "1",
            "guid": "E7065496-A5D2-EC11-9515-D8BBC1166DF4",
            "name": "Control Reader PR-x08 1",
            "_marker": "",
            "card_bits": "16",
            "card_start": "0",
            "fc_bits": "8",
            "fc_start": "16",
            "flags": "",
            "objname": "Control Reader PR-x08 1",
            "objtype": "PARSEC_PR_X08",
            "parent_id": "localhost",
            "parent_type": "SLAVE",
            "serial_number": ""
        }
    ]
}
```

16 Axxon Next integration

The configured integration of *Intellect* with *Axxon Next* allows you to receive video streams and events from *Axxon Next* cameras in *Intellect.*

ONVIF protocol is used for integration.

16.1 Integration with Axxon Next via ONVIF protocol

Axxon Next and *Intellect* integration is configured in several stages in the following order:

1. Update *Axxon Next* to the version 4.6.2 together with IP Drivers Pack version 3.74 (these are the minimum required versions) or later—see Axxon Next update, Updating Axxon Next Drivers Pack module, Updating Axxon Next on Linux OS.
2. In *Axxon Next,* open the port 8081. It is necessary for the ONVIF Server to work.
3. In *Axxon Next*, configure and start the ONVIF Server—see Configuring the ONVIF Server.
4. Update *Intellect* IP Drivers Pack to the minimum required version 3.74 or later. You can do this in two ways:
    a. Update the main product to the latest hotfix, and Drivers Pack will be updated together with it—see Updating Intellect, Hotfixes of Intellect and vertical solutions.
    b. Update Drivers Pack only—see Updating Drivers Pack separately from the main product.
5. In *Intellect,* configure the Video capture device in a certain way—see Configuring the Video capture device object for integration with Axxon Next via ONVIF.

As a result, the *Axxon Next* ONVIF Server will be connected to *Intellect* as a Video capture device with its corresponding video cameras that will be displayed as child devices.

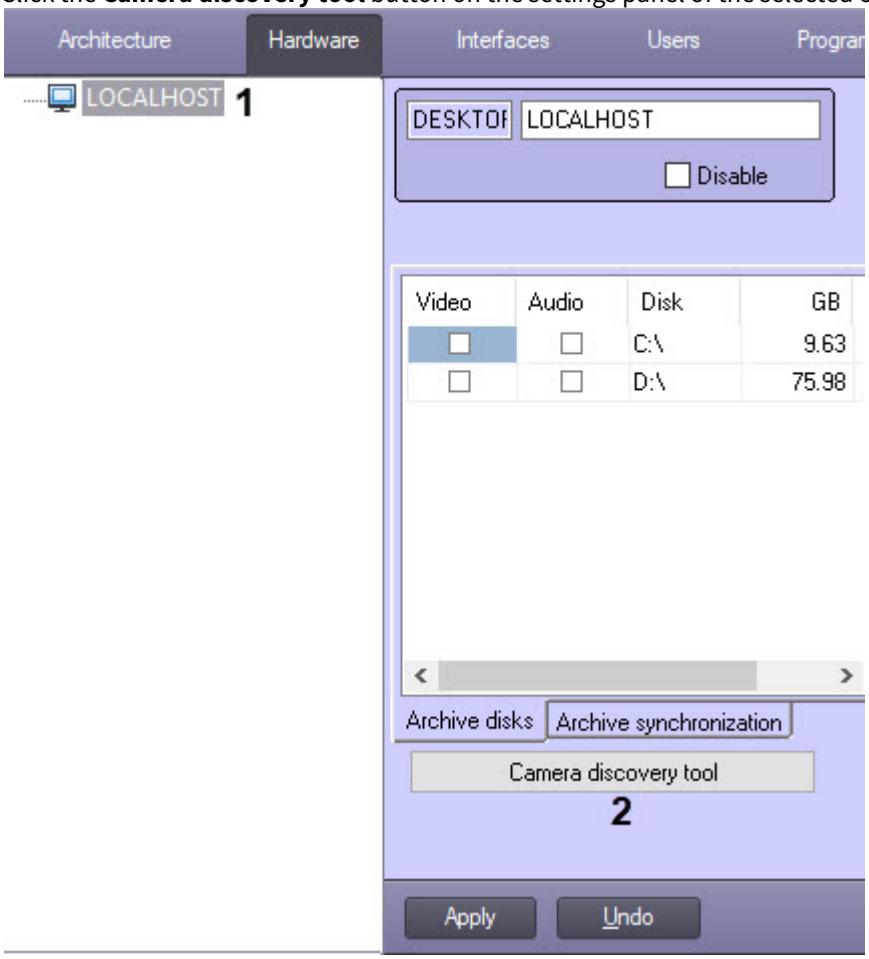
16.1.1 Configuring the Video capture device object for integration with Axxon Next via ONVIF

There are two options for configuring the **Video capture device** object to receive data from *Axxon Next*: using the **Camera discovery tool** (the preferred option) or creating an object in the **Hardware** objects tree. Both options are described below.

16.1.1.1 Using the Camera discovery tool

1. Go to the **Hardware** tab in the **System settings** dialog box. In the objects tree, select the **Computer** object corresponding to the Server, for which you want to configure the connection with *Axxon Next* (**1**).

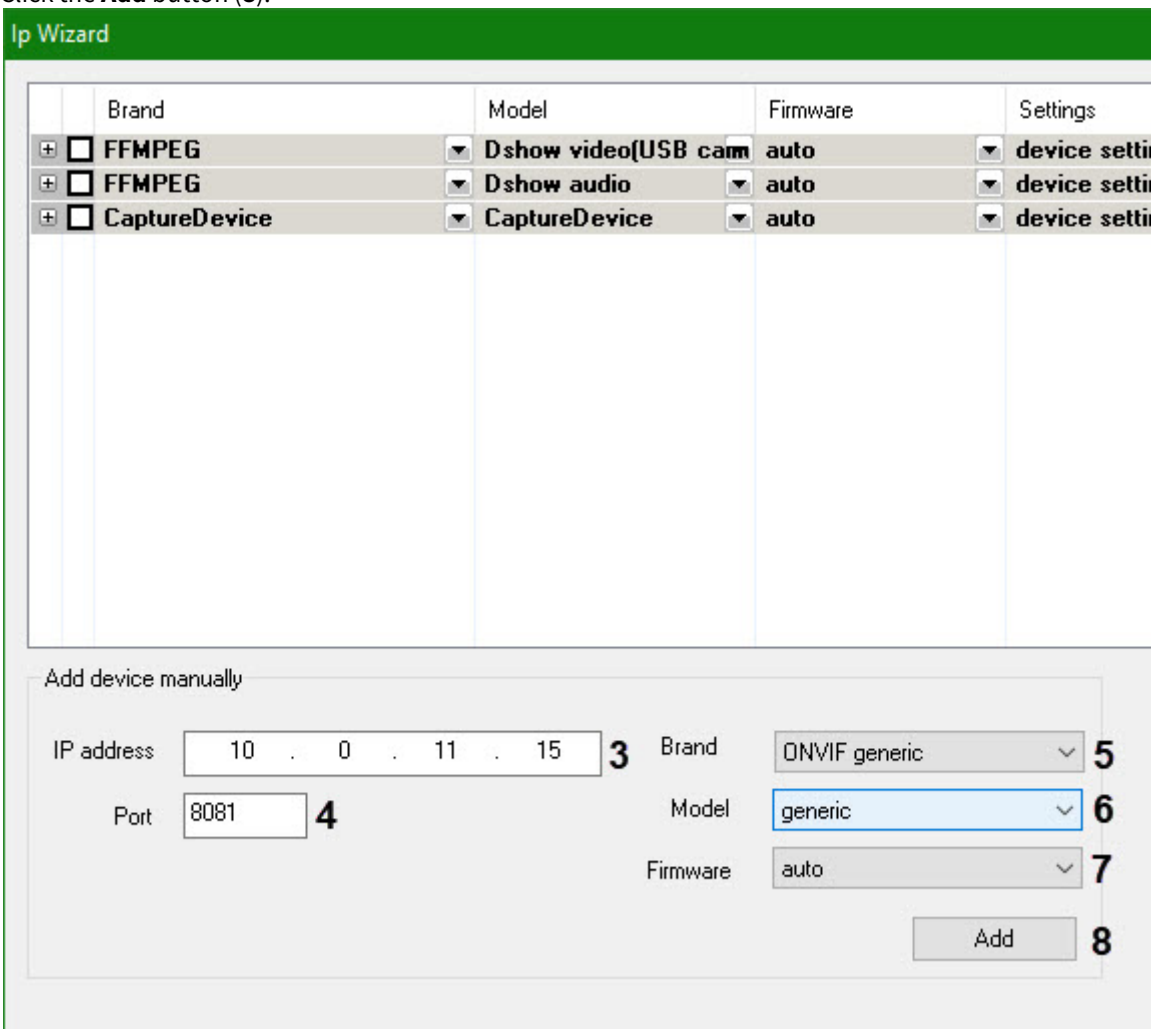
2. Click the **Camera discovery tool** button on the settings panel of the selected object (**2**).



As a result, the **Ip Wizard** window will open.

3. In the **Add device manually** section:
   a. In the **IP address** field, enter the IP address of the *Axxon Next* ONVIF Server (**3**).
   b. In the **Port** field, enter the port of the *Axxon Next* ONVIF Server (**4**).
   c. From the **Brand** drop-down list, select **ONVIF generic** (**5**).
   d. From the **Model** drop-down list, select **generic** (**6**).
   e. From the **Firmware** drop-down list, select **auto** (**7**).

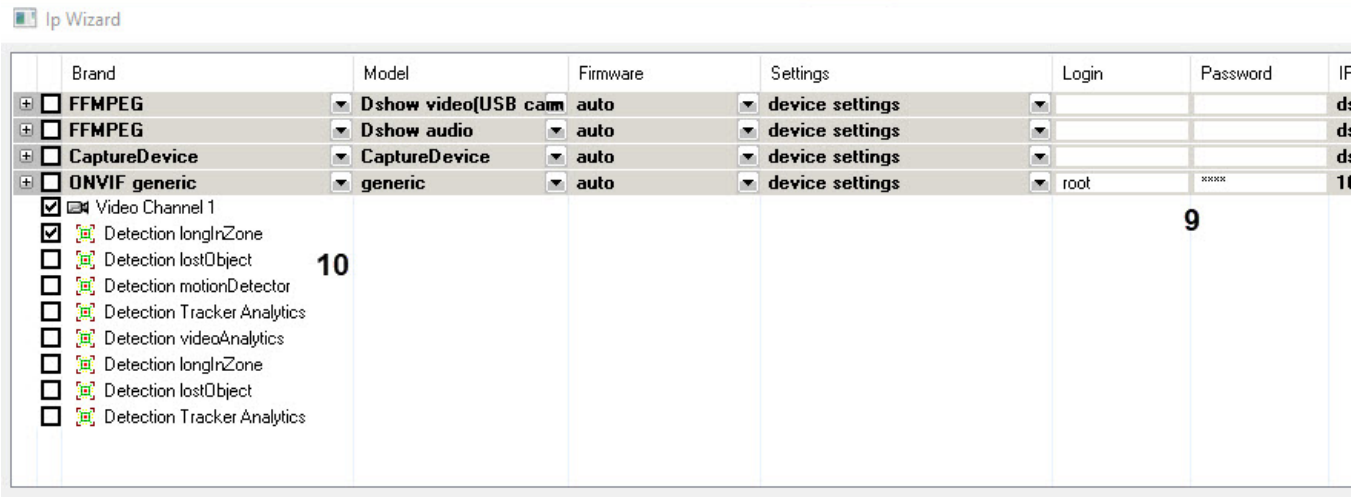
f. Click the **Add** button (**8**).



As a result, the IP device with the specified parameters will be displayed in the upper part of the window.

> **Note**
>
> In *Intellect* version 4.9, configuration is different: you need to select the **ONVIF** brand and the **4_channel_device** model.

4. In the **Login** and **Password** columns, enter the login and password to connect to *Axxon Next* (**9**).



5. Select the objects that you want to add to *Intellect.* To do this, expand the objects list by clicking the ⊞ button and set the checkboxes next to the objects that need to be created (**10**).

> **Note**
>
> In *Intellect* version 4.9, the **onvif_events** will be displayed as *Axxon Next* detection tools. You need to select them.

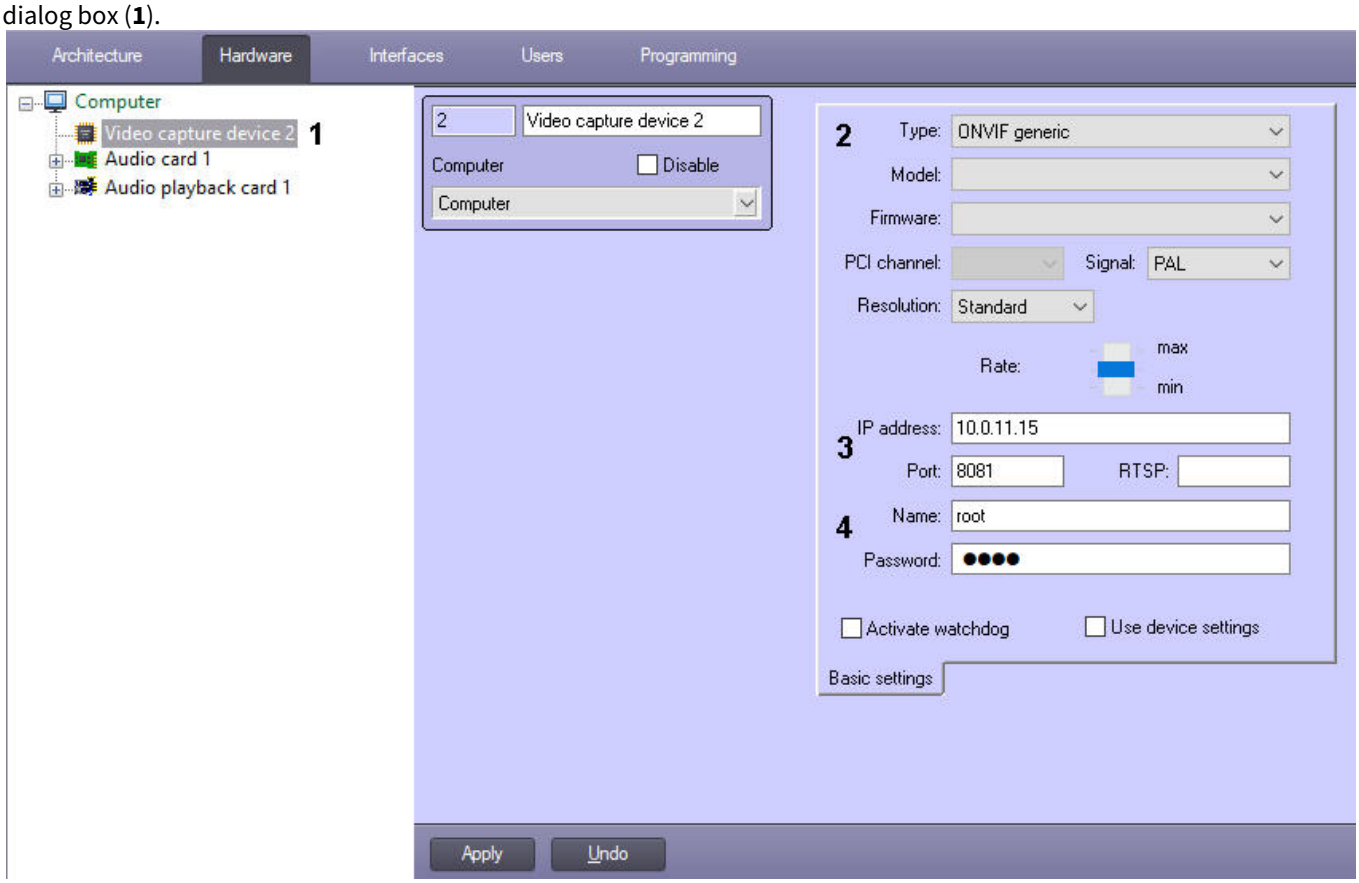
6. Click the Apply button.

The selected objects will be automatically created in the objects tree on the **Hardware** tab.

For a full description of the **Ip Wizard** settings, see Camera discovery tool.

16.1.1.2 Creating an object in the Hardware objects tree

1. Create the **Video capture device** object on the basis of the **Computer** object on the **Hardware** tab of the **System settings** dialog box (**1**).



2. From the **Type** drop-down list, select **ONVIF generic** (**2**).

> **Note**
>
> Earlier versions of *Intellect* may have the **ONVIF** or **Onvif(h264)** types. They can also be selected.

3. In the **IP address** and **Port** fields, enter the IP address and port of the *Axxon Next* ONVIF Server (**3**).
4. In the **Name** and **Password** fields, enter the name and the password to connect to *Axxon Next* (**4**).
5. Click the Apply button.

The **Video capture device** object created this way will correspond to the *Axxon Next* Server. It will automatically display the available child objects that are present in *Axxon Next*.

For a full description of the **Video capture device** object settings, see The Settings panel of the Video capture device object.

17 Configuring RabbitMQ

Intellect can act as receiver and transmitter of RabbitMQ events. Download and install the following components from the https://www.rabbitmq.com/install-windows.html website before using this functionality:

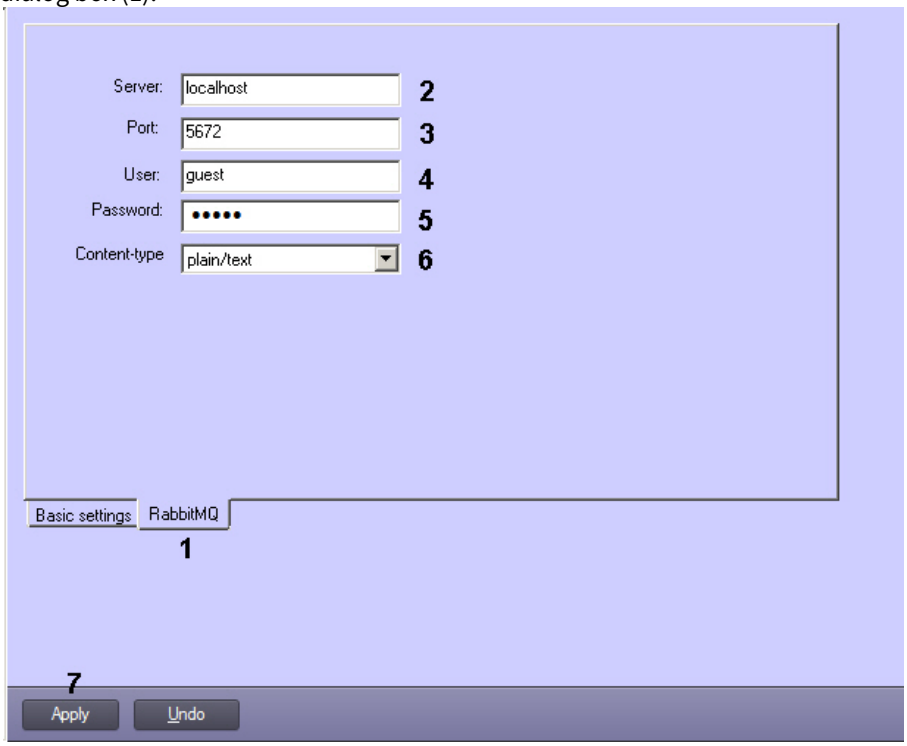
1. Rabbit service
2. Erlang

By default, RabbitMQ has a built-in **guest/guest** account that only works with **localhost**. Create a new account to use RabbitMQ remotely.

The examples below mention the amqp_sendstring.exe and amqp_listen.exe utilities. These utilities for processing and sending messages from the *Intellect* are to be implemented by programmer.

The connection to RabbitMQ is configured as follows:

1. Go to the **RabbitMQ** tab on the **Security zone** object settings panel on the **Programming** tab of the **System settings** dialog box (1).



2. In the **Host** field, enter the name or address of the RabbitMQ host (2).
3. In the **Port** field, enter the port to connect with the RabbitMQ host (3). If the port is 0 or not set (by default), the RabbitMQ client does not run.
4. In the **User** field, enter the RabbitMQ user name (4).
5. In the **Password** field, enter the RabbitMQ user password (5).
6. From the Content-type drop-down list, select the data type to be sent and received by Intellect (6):
    a. **application/json** – a message is sent and received in JSON format. If *Intellect* cannot parse the message in JSON format, an attempt is made to recognize the message in plain/text format.
    b. **plain/text** – the *Intellect* event is sent and received in the format "TYPE|ID|EVENT"
7. Click **Apply** (7).

17.1 Intellect provider and third party receiver

Intellect provider operates in amq.topic mode, and the receiver operates in amq.direct mode.

amq.topic allows for subscriptions ranging. Each event sent from *Intellect* is signed with a special header, which is built according to the following scheme:

routingkey = "intellect.event." + msg.GetSourceType() +"." + msg.GetSourceId() + "." + msg.GetAction();

So the receiver can subscribe to any event(s).

Example.

Subscription to all events:

```
amqp_listen.exe localhost 5672 amq.topic intellect.event.#
```

Subscription to Action==”RUN” events

```
amqp_listen.exe localhost 5672 amq.topic intellect.event.*.*.RUN
```

Important!

Events enter the RabbitMQ queue from each *Intellect* core individually. For example, if there are two cores in the system, and the event occurred on a core that has incorrect settings or does not have a connection to RabbitMQ, the event will not get into the queue, despite the fact that the second core will receive it. Each core sends only its own messages.

17.2 Intellect core receiver

The receiver is implemented according to the amq.direct scheme. It is subscribed to all events with the "bindingkey" key. bindingkey should have the following format: "intellect." + ComputerName (case sensitive).

When sending, the bindingkey must be specified exactly as it is registered at the receiver, otherwise the message will not be delivered.

Example events that will be receiver by Intellect core:

text:

```
amqp_sendstring.exe localhost 5672 amq.direct intellect.ASUS "CAM|1|HELLO"
```

or JSON

```
amqp_sendstring.exe localhost 5672  amq.direct intellect.ASUS {\"Type\":\"MACRO\",\"Id\":
\"1\",\"Action\":\"RUN\",\"Params\":{\"test1\":\"+++\",\"test2\":\"000\"}}
```

18 Intellect software Integration Guide. Postscript

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

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

If 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.

19 APPENDIX 1. DDI file structure

The table below contains the description of the fields of the table from the **Names** tab (the **<Objects>** section).

Field	Description
Name (<ObjectName>)	Object ID name
Visible name (<VisibleName>)	Visible name
Group name (<GroupName>)	The name of a group of objects. Used for grouping objects in *Intellect* settings tree

The table below contains the description of the fields of the table from the **Events** tab (the **<Events>** section).

Field	Description
Name (<EventName>)	Event ID name
Description (<EventDescription>)	Event description displayed in the event log
Event handling (<EventType>)	Used to set the background color in the event log: normal – no background color; alarm – red window; information – blue window
Sound support (<IsSoundEnabled>)	A sound file is played when a message is received
Do not send over network (<IsNetworkDisabled>)	Messages will not be sent over the network
Do not log (<IsProtocolDisabled>)	Message will not be displayed in the event log, and event will not be recorded in the database
Windows log (<IsWindowsLogEnabled>)	Record message in the Windows log *Note: Recording in the Windows log is impossible, if the event is not logged*

The table below contains the description of the fields of the table from the **Reacts** tab (the **<Reacts>** section).

Field	Description
Name (<ReactName>)	Reaction name
Description (<ReactDescription>)	The reaction description displayed in the context menu after a right-click on the object icon on the *Map*
Flags (<IsReactArm>)	Indication whether the reaction is performed either for a single object or for a group of objects from the same section

The table below contains the description of the fields of the table from the **Icons** tab (the **<Icons>** section).

Field	Description
Filename (<FileName>)	The part of the bmp file name that serves as an image ID. An image ID allows you to use multiple bmp files to show objects of the same type on the *Map* (see Using the ddi.exe Tool to Work with DDI files)
Name (<IconName>)	The description of the object bmp file

The table below contains the description of the fields of the table from the **States** tab (the **<States>** section).

Field	Description
Name (<StateName>)	State name
Image (<ImgName>)	The part of the bmp file name that serves as a state ID (see Using the ddi.exe Tool to Work with DDI files) *Note: The Map may display objects using lines, i.e. without using the bmp files. In this case, if an object changes its state, the line color changes. The color (RGB) of a state is set as follows:* ***<State>$R:G:B***
Description (<StateDescription>)	State description
Flashing (<IsStateFlashing>)	Display on the *Map*: normal – the icon does not flash, alarm – the icon flashes on the *Map*

The table below contains the description of the fields of the table from the **Transition Rules** tab (the **<Rules>** section).

Field	Description
Event (<EventName>)	The event that triggers the transition
Transition From (<FromStateName>)	The start state from which the transition is made
Transition To (<ToStateName>)	The end state to which the transition is made

20 APPENDIX 2. NissObjectDLLExt and CoreInterface class declarations

On the page:

20.1 CoreInterface

```
class CoreInterface

{

public:

                virtual BOOL DoReact    (React&) = 0;

                virtual BOOL NotifyEvent(Event&) = 0;

                virtual void SetupACDevice(LPCTSTR objtype, LPCTSTR objid, LPCTSTR
 objtype_reader) = 0;



                virtual   BOOL    IsObjectExist(LPCTSTR objtype, LPCTSTR id) = 0;

                virtual   BOOL    IsObjectDisabled(LPCTSTR objtype, LPCTSTR id) = 0;

                virtual Msg FindPersonInfoByCard(LPCTSTR facility_code, LPCTSTR card) = 0;

                virtual Msg FindPersonInfoByExtID(LPCTSTR external_id) = 0;

                virtual   CString GetObjectName (LPCTSTR objtype, LPCTSTR id) = 0;

                virtual   CString GetObjectState(LPCTSTR objtype, LPCTSTR id) = 0;

                virtual   void       SetObjectState(LPCTSTR objtype, LPCTSTR id, LPCTSTR state)
= 0;

                virtual   BOOL    IsObjectState(LPCTSTR objtype, LPCTSTR id, CString state) =
0;



                virtual   CString GetObjectParam    (LPCTSTR objtype, LPCTSTR id, LPCTSTR
 param) = 0;

                virtual   int  GetObjectParamInt (LPCTSTR objtype, LPCTSTR id, LPCTSTR param) =
0;
```

```
                virtual   CMapStringToStringArray* GetObjectParamList(LPCTSTR objtype, LPCTSTR
 id, LPCTSTR param) = 0;

                virtual   CStringArray*   GetObjectParamList(LPCTSTR objtype, LPCTSTR id,
LPCTSTR param, LPCTSTR name) = 0;

                virtual   void       GetObjectParams   (LPCTSTR objtype, LPCTSTR id, Msg& msg)
= 0;

                virtual   void       SetObjectParamInt (LPCTSTR objtype, LPCTSTR id, LPCTSTR
 param, int val) = 0;

                virtual   CString GetObjectIdByParam(LPCTSTR type, LPCTSTR param, LPCTSTR val)
= 0;

                virtual   CString GetObjectIdByName(LPCTSTR type, LPCTSTR name) = 0;

                virtual   CString GetObjectParentId(LPCTSTR objtype, LPCTSTR id, LPCTSTR
 parent) = 0;

                virtual   int GetObjectIds(LPCTSTR objtype, CStringArray& list, LPCTSTR main_id
= NULL) = 0;

                virtual   int GetObjectChildIds(LPCTSTR objtype, LPCTSTR objid, LPCTSTR
 childtype, CStringArray& list) = 0;

};
```

20.2 NissObjectDLLExt

```
class NissObjectDLLExt

{

protected:

                CoreInterface* m_pCore;

public:

                NissObjectDLLExt(CoreInterface* core) { m_pCore = core; }



                virtual CString GetObjectType() = 0;

                virtual CString GetParentType() = 0;

                virtual int GetPos()  { return -1; }

                virtual CString GetPort() { return CString(); }

                virtual CString GetProcessName() { return CString(); }

                virtual CString GetDeviceType() { return CString(); }
```

```
        virtual BOOL HasChild() { return FALSE; }
        virtual UINT HasSetupPanel()  { return FALSE; }
        virtual void OnPanelInit(CWnd*) {}
        virtual void OnPanelLoad(CWnd*,Msg&) {}
        virtual void OnPanelSave(CWnd*,Msg&) {}
        virtual void OnPanelExit(CWnd*) {}
        virtual void OnPanelButtonPressed(CWnd*,UINT) {}
        virtual BOOL IsRegionObject()   { return FALSE; }
        virtual BOOL IsProcessObject()  { return FALSE; }
        virtual BOOL IsIncludeParentId()             { return 0; }
        virtual BOOL IsWantAllEvents()                                  { return 0; }
        virtual CString DescribeSubscribeObjectsList() { return CString(); }
        virtual CString GetIncludeIdParentType(){ return CString(); }
        virtual CString DescribeParamLists(){ return CString(); }

        virtual void OnCreate(Msg&) {}
        virtual void OnChange(Msg&,Msg&) {}
        virtual void OnDelete(Msg&) {}
        virtual void OnInit(Msg&)           {}
        virtual void OnEnable(Msg&)  {}
        virtual void OnDisable(Msg&) {}
        virtual BOOL OnEvent(Event&) { return FALSE; }
        virtual BOOL OnReact(React&) { return FALSE; }
};
```