Hardware and Software Module Integration

PSIM 1.0 (english)

Last update 08/18/2022

Table of Contents

1 Axxon PSIM Integration Developer Kit (IIDK)

1.1 Examples of Managing System Objects

1.1.1 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

1.1.2 Obtaining info on core queues with GET_QUEUE_INFO command

Use the GET_QUEUE_INFO command to request info about the queues in the Axxon PSIM 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.

1.1.3 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**

1.1.4 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.

1.1.5 Live and archived video

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 Axxon PSIM Axxon PSIM Demo kit available for download at the page of Axxon PSIM Software Integration Guide (HTTP API, IIDK, ActiveX).

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)

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>
```

1.1.6 Showing Information Messages. SET_STATE

To show an information message on the display of Axxon PSIM'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<>

1.1.7 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).

1.1.8 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<>,galaxy_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_duress<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_access<0>,
pnet3_block<0>,passport<>,person<>,galaxy_menu_choice<0>,flags<0>,auto_number<>,phone<>,pin<>,rubeg8_AccessToBCPTimeZoneNumber<>,
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_2cards_mask<0>

Note.

User data can also be received via direct request to *Axxon PSIM* 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 *Axxon PSIM* software database operation in Administrator's Guide, the Appendix 4. Axxon PSIM™ software database management.

1.1.9 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>

1.1.10 Determining Computers Where Axxon PSIM was Unloaded (via Port 1030)

If Axxon PSIM 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<psim_host.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 *Axxon PSIM* was unloaded and the date and time

1.1.11 Working with the System in the Multiuser Mode

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

If users have been created and access rights have been configured in Axxon PSIM, 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.

1.1.12 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.

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.

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>

1.2 Sent Message Syntax

1.2.1 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 Axxon PSIM (see Axxon PSIM'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
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);

1.2.2 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 Axxon PSIM
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.

1.2.3 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 Axxon PSIM Software Integration Guide (HTTP API, IIDK, ActiveX).

1.3 IIDK Functions

1.3.1 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.

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 *Axxon PSIM* Server

port	TCP/IP port over which the connection is established
id	SLAVE connection ID, for video
lpfunc	Callback function receiving messages from *Axxon PSIM*
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.

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.

IsConnected

IsConnected returns TRUE if the client is connected to server.

```
BOOL IsConnected();
```

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 *Axxon PSIM* Server
port	TCP/IP port over which the connection is *established*
id	Core connection ID, for video
lpfunc	Callback function receiving messages from *Axxon PSIM*
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

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

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.

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

SetPingTime

The function enables and sets the interval for sending KeepAlive messages to the *Axxon PSIM* 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.

1.3.2 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 Axxon PSIM, **DISCONNECTED** is passed to the callback function.

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

1.3.3 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	
LPCTSTR msg	Message text	

```
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


Disconnect (id);
```

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

1.3.4 Connect

To establish communication between a functional module and Axxon PSIM, 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 Axxon PSIM	

```
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);
}
```

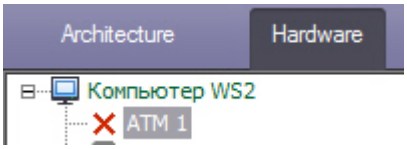
> **Note:**
>
> **Void _stdcall myfunc** is called in a separate stream (not in the application's main stream).

The developer handles received messages as needed.

1.4 Connecting to Axxon PSIM

1.4.1 Features of ATMs integration. ATM object

The **ATM** object can be used to send events from ATM software to *Axxon PSIM* 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.

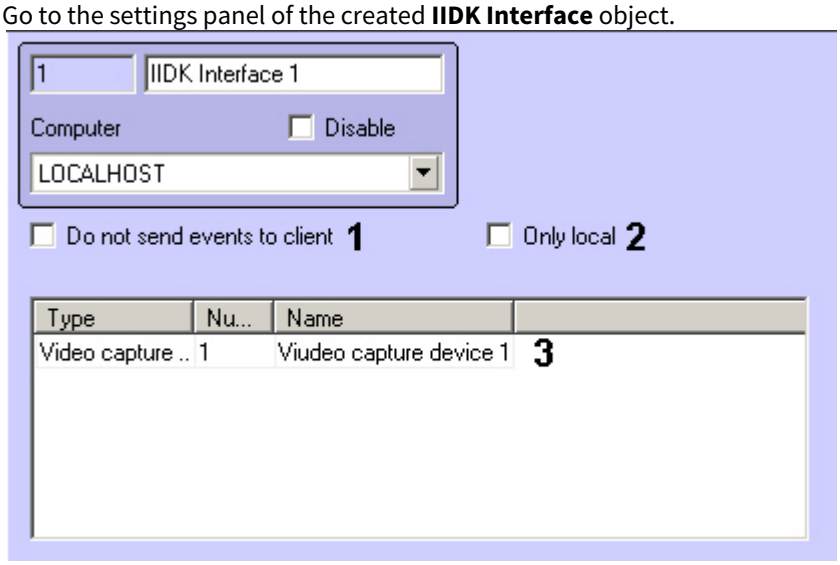
Connection method and message syntax for the **ATM** object are the same as those for the **IIDK Interface object**, though 1009 port is in use for sending messages (see also Connection Parameters and Message Syntax (port 900)).

1.4.2 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.

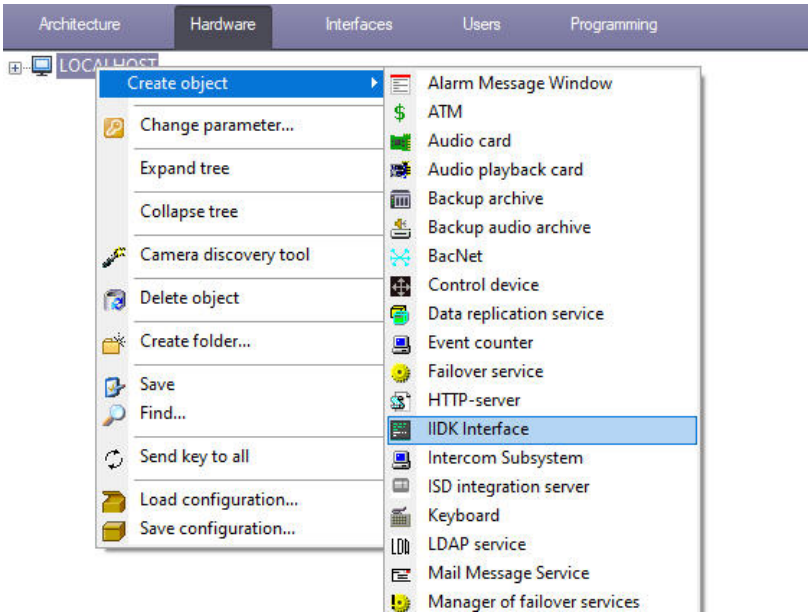
1.4.3 IIDK Interface Object

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 *Axxon PSIM* object tree.

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

> **Note**
>
> If *Axxon PSIM* is started in the Axxon PSIM 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 *Axxon PSIM* 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*.

1.4.4 Connection Parameters

Axxon PSIM'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 Axxon PSIM'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 *Axxon PSIM* settings.

> **Note.**
>
> If connection to the server (**IIDK Interface object**) from remote computer is required, then there is no need to install *Axxon PSIM* on the remote computer, but this computer is to be added to *Axxon PSIM* 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 Axxon PSIM'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 "").

1.5 General Information on IIDK

1.5.1 IIDK Components

The *IIDK* includes the following development tools:

1. *iidk.ocx* – ActiveX control. When installing *Axxon PSIM* 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 <*Axxon PSIM* installation directory>\Tools folder.

1.5.2 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.

1.5.3 Purpose of the IIDK

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

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

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

2 Integrating hardware and software modules with Axxon PSIM

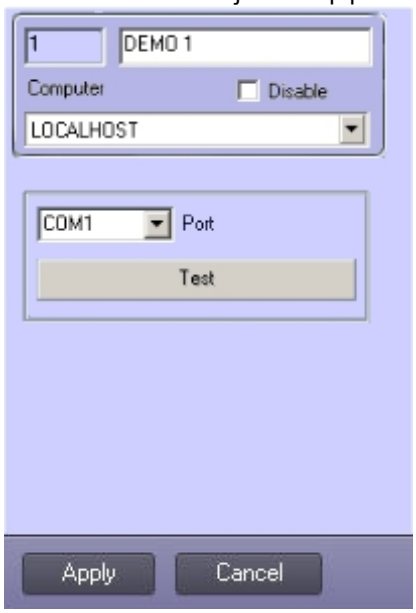
2.1 Creating and Configuring Integrated Objects (Modules) in Axxon PSIM

To create and configure an integrated object (module) in Axxon PSIM:

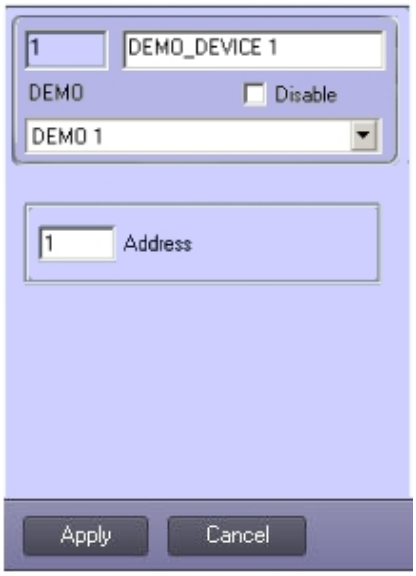
1. Copy the MDL and RUN files to the *Axxon PSIM\Modules* folder.
2. Start Axxon PSIM.
3. Under the **Computer** object, create the objects added earlier using the software module. For the *Axxon PSIM Demo* module, Under the **Computer** object, create the **Axxon PSIM Demo** object.

> **Note:**
>
> Under the **Axxon PSIM Demo** object, create a child object, **DEMO_DEVICE.**

As a result, the setup panels of the created objects become available.
Axxon PSIM Demo object setup panel:



DEMO_DEVICE object setup panel:



4. Set up the objects.

The integrated objects are now created and set up in Axxon PSIM.

2.2 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 *Axxon PSIM Integration Developer Kit (IIDK),* which is covered in detail in Section Axxon PSIM 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 *Axxon PSIM 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, "Axxon PSIM Demo|1|RESTORE"); // restore the connection with the Axxon
PSIM 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 Axxon PSIM'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*.

2.3 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 Axxon PSIM Demo object receives the following messages: Axxon PSIM Demo|1|UPDATE_CAM| parameters of the camera Axxon PSIM Demo|1|UPDATE_GRABBER| parameters of the video capture card

Method	Description	Example
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 "Axxon PSIM 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 *psim.sec* key file.
	Attention! This parameter must be set in consultation with AxxonSoft.
		`virtual int GetPos()`
		
		`{ return -1; }`
		*Note: If Axxon PSIM is run in the Axxon PSIM 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"; }`

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 "Axxon PSIM Demo"; }`

Method	Description	Example
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.	

Method	Description	Example

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");`
		`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.	

<table>
<tr><th>Method</th><th>Description</th><th>Example</th></tr>
<tr><td rowspan="2">virtual void OnPanelButtonPressed(CWnd*,UINT)</td><td>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</td><td><pre>Virtual void
 OnPanelButtonPressed (CWnd*
pwnd,UINT id)

{

if(id==IDC_TEST)

{ React react("Axxon PSIM
Demo",Id,"TEST");

m_pCore->DoReact(react); }

}</pre></td></tr>
<tr><td>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.</td><td><pre>HINSTANCE prev_hinst =
AfxGetResourceHandle();

HMODULE hRes =
GetModuleHandle("Axxon PSIM
Demo.mdl");

If (hRes)
AfxSetResourceHandle (hRes);

//Code for showing a dialog
box:

CXXXDialog dlg;

dlg.DoModal();

AfxSetResourceHandle(prev_hin
st);</pre></td></tr>
</table>

Method	Description	Example
virtual BOOL IsRegionObject()	Shows whether the object supports Axxon PSIM'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 Axxon PSIM'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 Axxon PSIM's panel when the object is disabled. The parameter is a reference to a message that contains object information.	

Method	Description	Example

virtual BOOL OnEvent(Event&)	Used to handle the events that are passed as the parameter.	virtual BOOL
		
		OnEvent(Event& event)
		
		{
		
		If
		
		(event.GetAction() == "ACCESS_IN" ||
		
		event.GetAction() == "ACCESS_OUT")
		
		{
		
		Msg per = m_pCore-> FindPersonInfoByCard(event.GetParam("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")

Method	Description	Example

		{
		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 NissObjectAxxon PSIM Demo(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.

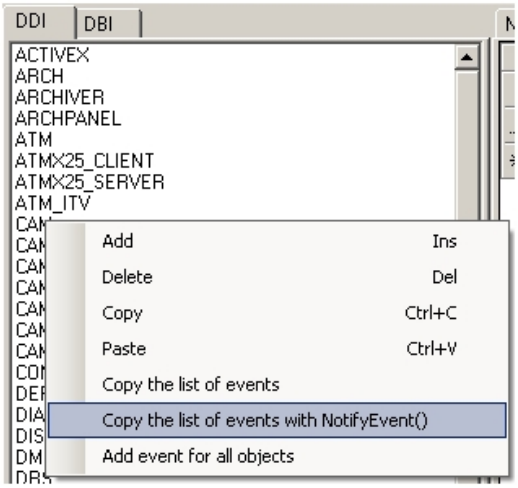
All object setup panels are stored in resources as dialogs. Each dialog ID has the format **IDD_object_SETUP**, where **object** is the name of the corresponding object. For example, the ID of the **Axxon PSIM Demo object** is **IDD_DEMO_SETUP**, and the ID of the **DEMO_DEVICE** object is **IDD_DEMO_DEVICE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

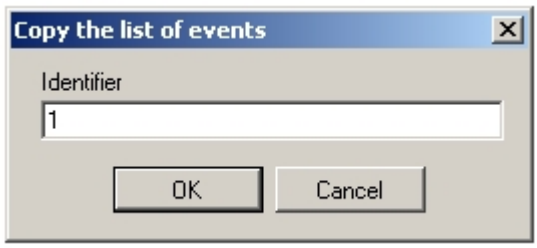
2.4 Additional Functionality of the ddi.exe Utility

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()**.



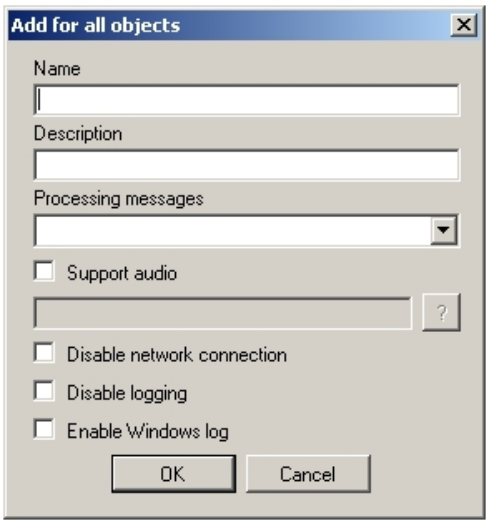
2. A dialog box opens. In the dialog box, enter the object ID to be used by the NotifyEvent function and click **OK.**



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**.

2.5 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 Axxon PSIM.ddi file contains the properties of Axxon PSIM's main objects. For your own objects, we recommend creating a separate file, Axxon PSIM.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 Axxon PSIM software package. On startup of the software package, the DDI files are merged.

If an object is duplicated in several ddi-files, then at Axxon PSIM 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 Axxon PSIM.xxx.ddi, Axxon PSIM.xxx1.ddi and Axxon PSIM.xxx2.ddi, the properties from the Axxon PSIM.xxx2.ddi will be applied.

2.5.1 Adding Object Information to Axxon PSIM.ddi

This section shows how to add information on the **Axxon PSIM Demo** object to Axxon PSIM.ddi by using a text editor.

To add information on the **Axxon PSIM Demo** object, do the following:

1. Go to *Axxon PSIM**Languages*\en folder and open the psim.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 **Axxon PSIM 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 Axxon PSIM.ddi file.

The information on the **Axxon PSIM Demo** object has now been added to Axxon PSIM.ddi.

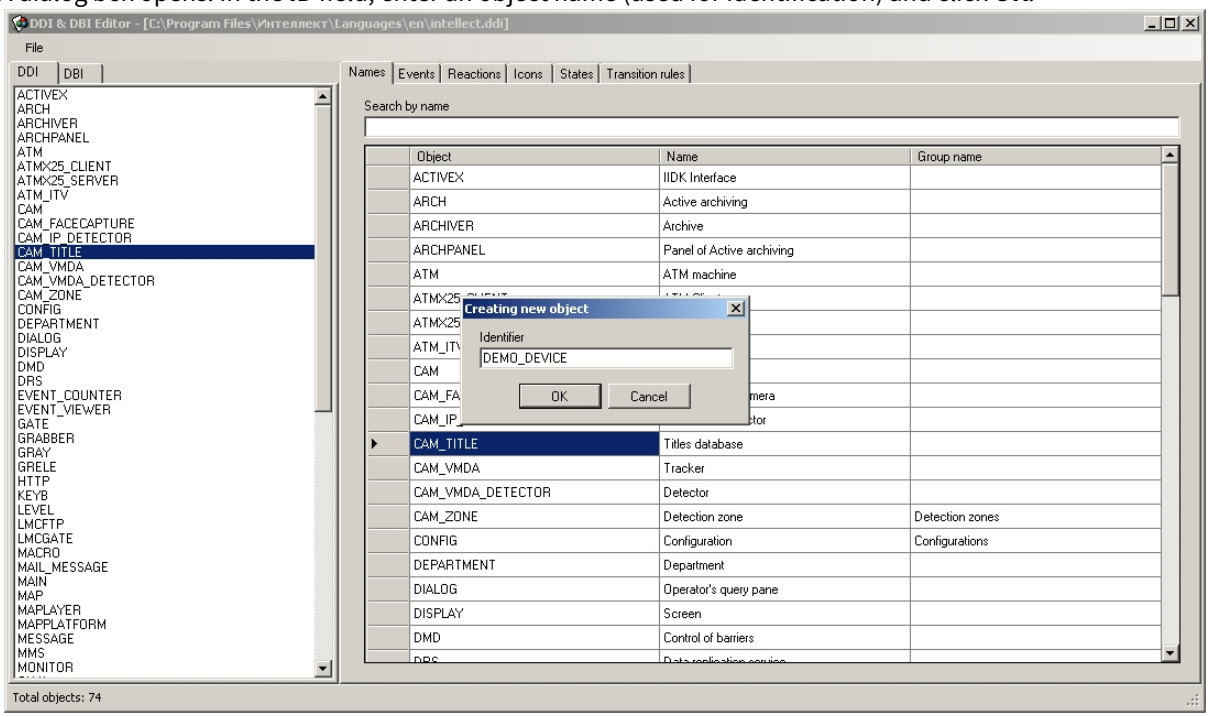
> **Attention!**
>
> After making changes to the DDI file, you must update the database structure by using idb.exe (see Steps 4–7 in Section Adding Objects to psim.dbi).

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

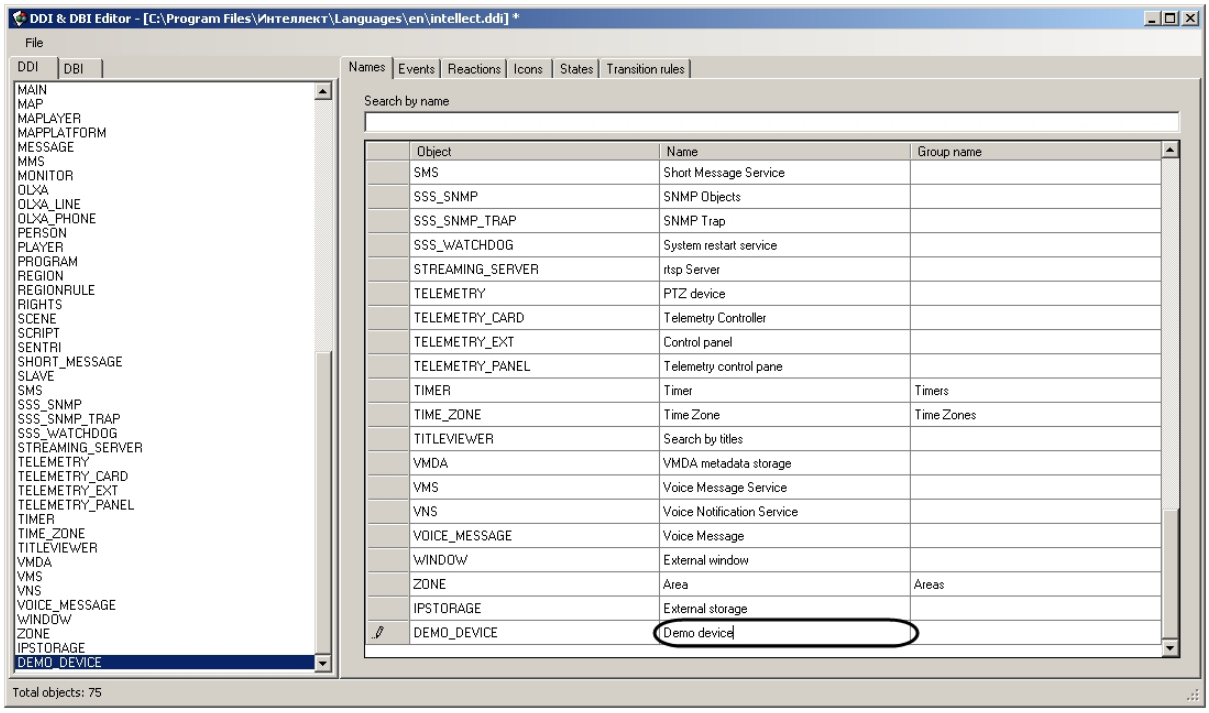
This section shows how to add information on the **DEMO_SERVICE** object to Axxon PSIM.ddi by using the *ddi.exe* tool.

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

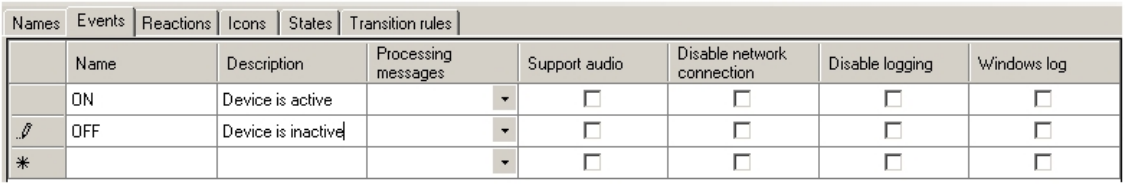
1. Open Axxon PSIM.ddi file in one of the following ways:
   a. Go to the *Axxon PSIM\Tools* folder and run *ddi.exe*.
      i. In the program window, select the **DDI** tab.
      ii. In the **File** menu, select **Open**. The **Open** dialog box appears.
      iii. Go to the *Axxon PSIM\Languages*\en folder and select Axxon PSIM.ddi. The window of *ddi.exe* shows a list of objects.
   b. Open the *Axxon PSIM\Tools* folder in Windows Explorer or any other file manager, then double-click the Axxon PSIM.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**.



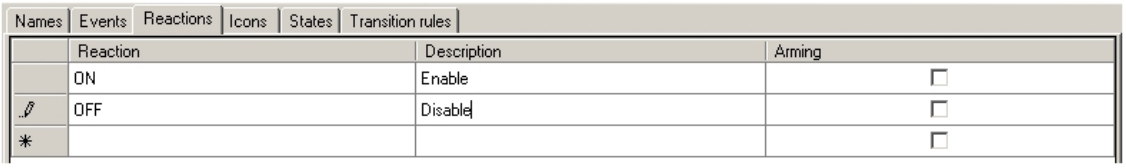
   The DEMO_DEVICE object is now shown in the list of objects.
4. In the **Names** tab, enter an object name.



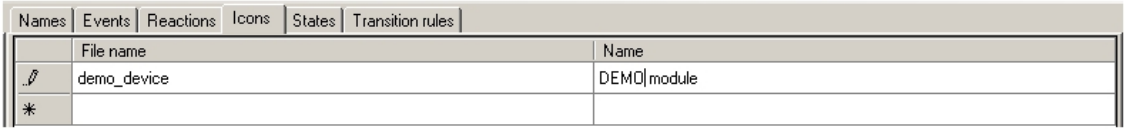
5. In the relevant tabs, add information on the DEMO_DEVICE object.
   a. In the **Events** tab, add the **ON** and **OFF** events.



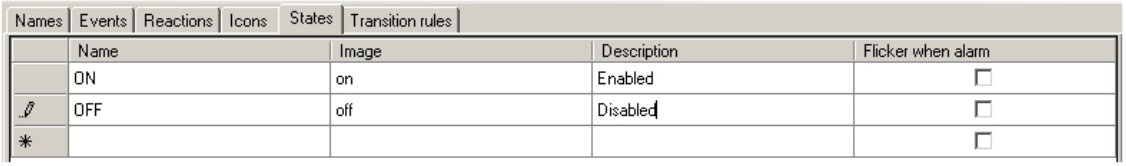
   b. In the **Reactions** tab, add the **ON** and **OFF** actions.



   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.**



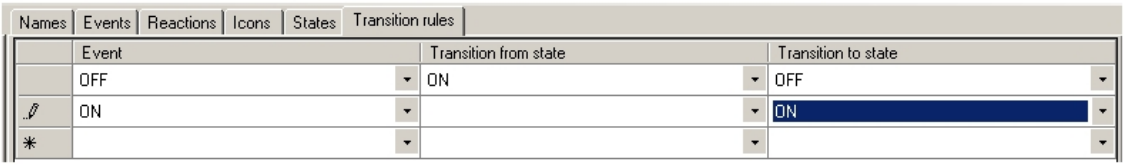
   d. In the **States** tab, add the **ON** and **OFF** states. To show an object state on the **Map,** enter a BMP file name (the part that serves as an ID of the state).



> **Note.**
>
> The names of the BMP files in the Axxon PSIM\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 psim.dbi).

2.6 Editing the DBI file

The psim.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 Axxon PSIM.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 Axxon PSIM software package. On startup of the software package, the DBI files are merged.

2.6.1 Adding Objects to Axxon PSIM.dbi

Objects are added to psim.dbi as follows:

1. Go to *Axxon PSIM's* root folder and open the psim.dbi file with a text editor.

2. Add the objects to psim.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 *Axxon PSIM.*

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.

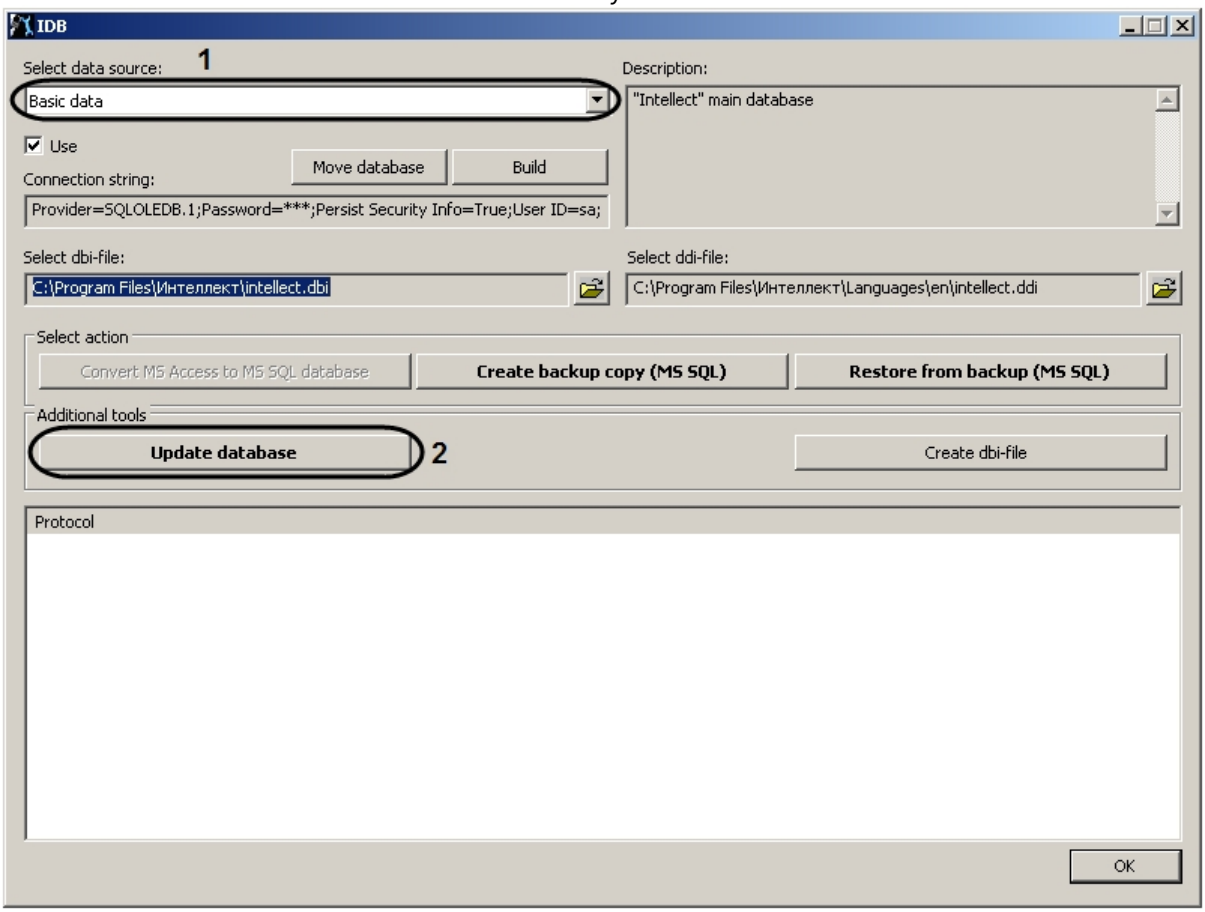
Beside the mandatory fields, the objects of the Axxon PSIM Demo module contain the following fields:

a. **port** – COM port address;
b. **address** – device address.

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

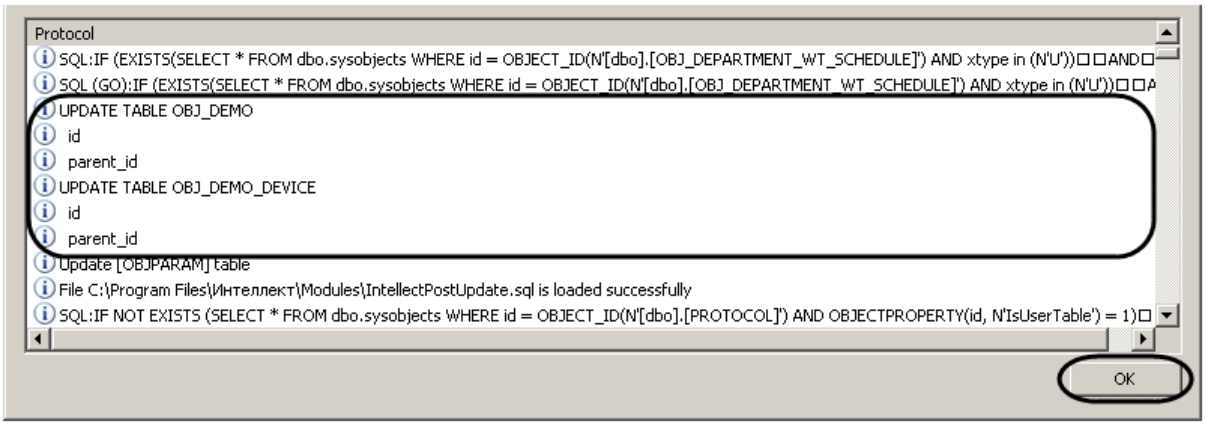
```
[OBJ_DEMO]
id, CHAR, 16
name, CHAR, 60
parent_id, CHAR, 16
flags, INTEGER
port, CHAR, 5

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

3. Save the changes to the psim.dbi file.
4. Go to Axxon PSIM'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 *Axxon PSIM's* configuration database.

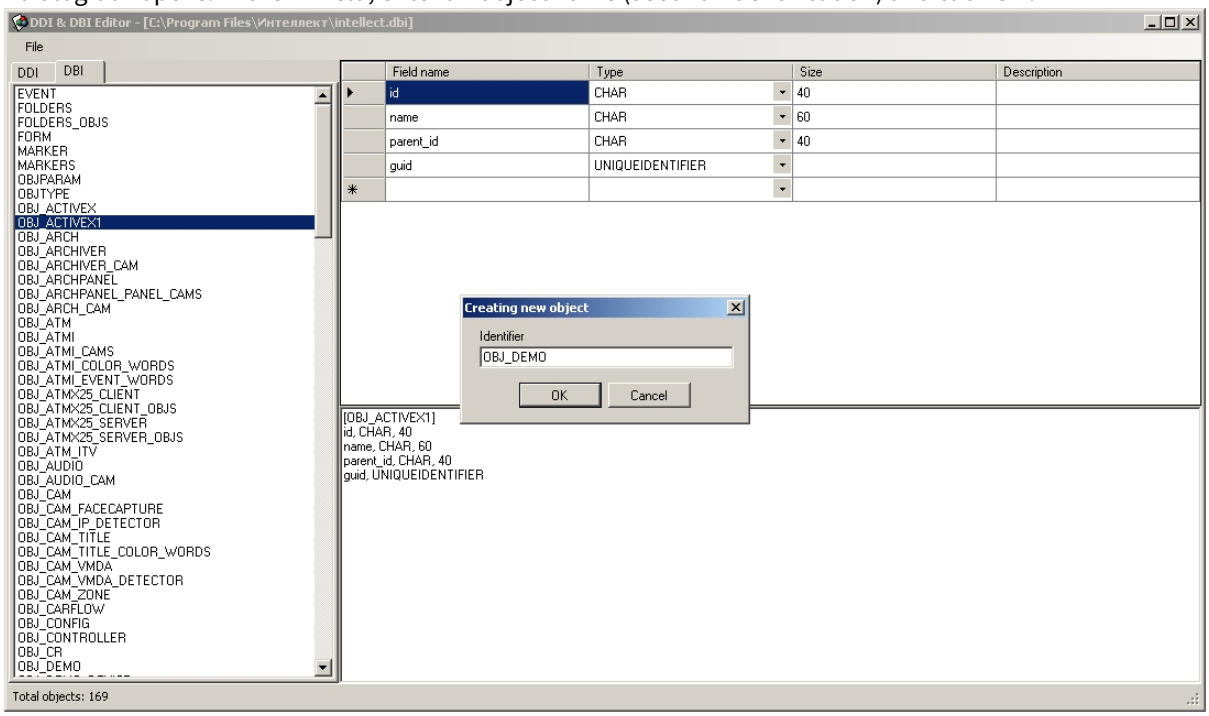
2.6.2 Using the ddi.exe Tool to Work with DBI files

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

1. Go to the *Axxon PSIM\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 Axxon PSIM's root folder and select the psim.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.**



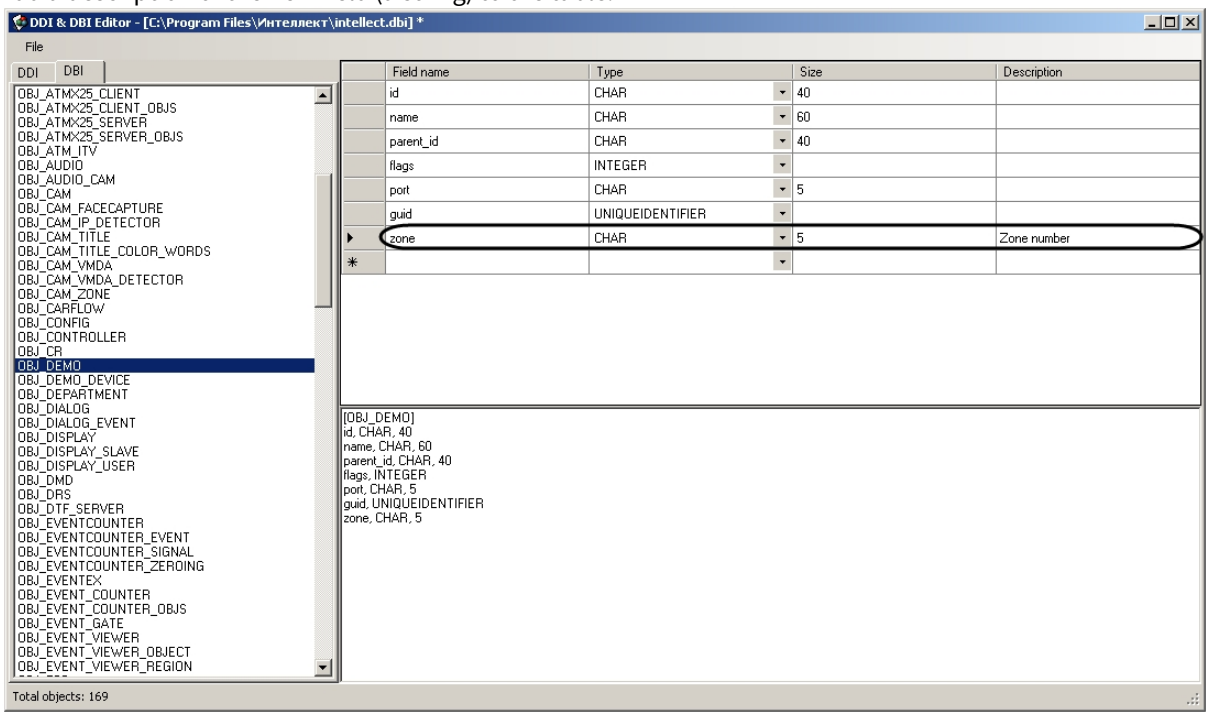
> **Note:**
>
> The mandatory fields are automatically added to the created object (see Section Adding Objects to psim.dbi).

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 psim.dbi).

2.7 General information on hardware and software modules integrating

To integrate a hardware and software (functional) module into Axxon PSIM, 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 *Axxon PSIM\Modules* folder.

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 Axxon PSIM'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 *Axxon PSIM Demo* module, which emulates working with virtual hardware. This module includes devices with unique addresses for accessing and polling these devices. Thus the system includes a configuration consisting of 2 main objects: a parent object, **Axxon PSIM Demo,** with the **COM port** parameter, and a child object, **DEMO_DEVICE**, with the **Address** parameter. The system allows you to perform a number of actions with devices and to pass all their events to the system core.